НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ» ІМЕНІ ІГОРЯ СІКОРСЬКОГО
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

«До захисту допущено»
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2019р.

Магістерська дисертація

зі спеціальності - 122 Комп’ютерні науки

за спеціалізацією - Комп’ютерний моніторинг та геометричне моделювання процесів і систем

на тему

Моделювання процесів міжагентної взаємодії в мережах Smart Grid

Виконав: студент 6 курсу, групи ТР-81мп

Швайка Дмитро Андрійович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник

доц. к.ф.-м.н. Тарнавський Юрій Адамович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент

_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
„Київський політехнічний інститут” імені Ігоря Сікорського

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти другий, магістерський

зі спеціальності - 122 Комп’ютерні науки

за спеціалізацією - Комп’ютерний моніторинг та геометричне моделювання процесів і систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
Коваль О.В.

_____
(прізвище, ініціали) (підпис)

«____» _____

2019р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ СТУДЕНТУ

Швайці Дмитру Андрійовичу
(прізвище, ім’я, по батькові)

1. Тема дисертації Моделювання процесів міжагентної взаємодії в мережах Smart Grid

Науковий керівник Тарнавський Юрій Адамович, доцент, доц. к.ф.-м.н.
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”_____ 2019 р. №

2. Строк подання студентом роботи

3. Об’єкт дослідження: комп’ютерні інформаційні системи і технології розумного будинку

4. Предмет дослідження: системи інтелектуального аналізу даних в умовах розумного будинку

5. Перелік питань, які потрібно розробити

- проаналізувати протокол керування пристроями;
- спроектувати архітектуру системи для моделювання процесів міжагентної взаємодії в мережах Smart Grid;
- розробити програмне забезпечення для моделювання процесів міжагентної взаємодії в мережах Smart Grid.

6. Орієнтований перелік ілюстративного матеріалу: архітектура системи, графічне представлення інтерфейсу, схема роботи сервера, приклади роботи програмного модулю.

7. Орієнтований перелік публікацій _____

1. XVII Міжнародна науково-практична конференція молодих вчених та студентів “Сучасні проблеми наукового забезпечення енергетики” (м. Київ, 23-26 квітня 2019 року).

8. Дата видачі завдання ”10” вересня 2018р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Відмітки
1.	Отримання завдання	10.09.2018р.	
2.	Аналіз вимог завдання, вибір методів і засобів розв’язання поставленої задачі	01.10.2018р. — 31.01.2019р.	
3.	Підготовка матеріалів магістерської роботи	01.10.2018р. — 20.11.2019р	
4.	Підготовка публікацій	01.03.2018р. — 26.04.2019р.	
5.	Захист програмного продукту	24.10.2019р	
6.	Передзахист	20.11.2019р	
7.	Захист	.12.2019р	

Студент _____ Швайка Д.А.
(підпис) (прізвище та ініціали,)

Науковий керівник _____ Тарнавський Ю.А.
(підпис) (прізвище та ініціали,)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: «Моделювання процесів міжагентної взаємодії в мережах Smart Grid»

студентом: Швайка Дмитром Андрійовичем

Магістерська дисертація складається зі вступу, шести розділів, висновку, переліку посилань з 32 найменувань, 1 додаток, і містить 31 рисунок, 9 таблиць. Повний обсяг магістерської дисертації складає 66 сторінок, з яких перелік посилань займає 3 сторінки, додатки – 1 сторінка.

Актуальність теми. Автоматизація є важливим напрямком розвитку підприємств та практики управління бізнес-процесами. Вона забезпечує ефективність роботи завдяки програмно-апаратним системам, які збільшують швидкість виконання та зменшують виробничі затрати. Інтернет речей в свою чергу забезпечує величезні апаратні та програмні можливості для реалізації проектів незалежно від сфери застосування.

Тому актуальним є дослідження можливостей Smart Grid, зокрема інтернету речей, в поєднанні з хмарними сервісами, зокрема, як ефективного рішення для автоматизації процесів в сфері торгівлі, так і в якості уніфікованого рішення для автоматизації процесів в будь-якій галузі або “розумному будинку”.

Метою дослідження є створення програмного забезпечення для автоматизації у будинку та для керування iot пристроями з iOS пристроїв.

Для досягнення поставленої задачі були сформульовані наступні **завдання дослідження**, що визначили логіку дослідження та його структуру:

- проаналізувати протокол керування пристроями;
- спроектувати архітектуру системи для моделювання процесів міжагентної взаємодії в мережах Smart Grid;
- розробити програмне забезпечення для моделювання процесів міжагентної взаємодії в мережах Smart Grid.

Об’єктом дослідження є комп’ютерні інформаційні системи і технології розумного будинку.

Предметом дослідження є системи інтелектуального аналізу даних в умовах розумного будинку.

Методи дослідження. Розв’язання поставлених задач виконувались з використанням наступних методів:

- метод аналізу;
- метод системного аналізу;
- метод порівняння;
- метод проектування логічних структур даних;
- метод логічного узагальнення результатів.

Наукова новизна одержаних результатів. Найбільш суттєвими науковими результатами магістерської дисертації є:

- удосконалення методу розпізнавання активності на базі даних сенсорів у реальному часі;
- розробка модулю для взаємодії з пристроями, яки підтримують Apple HomeKit;
- розробка програмної систему яка використовуючи розроблену модель здійснює керування пристроями за створеному сценаріями.

Практичне значення одержаних результатів полягає в можливості використання програмної системи для автоматизації розумних будинків а також ручному керуванні пристроями з сумісних з Apple HomeKit гаджетів.

Ключові слова. *МОДЕЛЮВАННЯ, APPLE HOMEKIT, АВТОМАТИЗАЦІЯ, РОЗУМНИЙ БУДИНОК.*

ABSTRACT

on master's thesis

on topic: «Modeling interagency interaction processes in Smart Grid networks »

Student: Dmytro Shvaika

The master's thesis consists of an introduction, six sections, conclusion, a list of references of 32 names, 1 appendix, and contains 31 figures, 9 tables. The full volume

of the master's thesis is 66 pages, of which the list of links occupies 3 pages, the annexes - 1 page.

Topicality. Automation is an important area of enterprise development and business process management practices. It delivers performance through software and hardware systems that increase execution speed and reduce production costs. The Internet of Things, in turn, provides enormous hardware and software for project implementation, regardless of scope.

Therefore, it is relevant to explore the capabilities of Smart Grid, in particular the Internet of Things, in combination with cloud services, in particular, as an effective solution for automation of trade processes, and as a unified solution for automation of processes in any industry or "smart home".

The aim of the research is to create software for home automation and control of iot devices from iOS devices.

To achieve this task, the following **research objectives** were formulated, which defined the logic of the study and its structure:

- analyze the device management protocol;
- to design the system architecture for modeling the processes of interagency interaction in Smart Grid networks;
- to develop software for modeling interagency interaction processes on Smart Grid networks.

The subject of the study is computerized information systems and smart home technologies.

The subject of the study is data mining systems in a smart home.

Research methods. The tasks were solved using the following methods:

- method of analysis;
- system analysis method;
- method of comparison;
- method of designing logical data structures;
- method of logical generalization of results.

Scientific novelty of the obtained results. The most significant scientific results of the master's thesis are:

- improvement of the method of recognition of activity on the basis of sensors in real time;
- developing a module to interact with devices that support Apple HomeKit;
- development of a software system that, using the developed model, manages the devices in the created scripts.

The practical value of the results is the ability to use a software system to automate smart homes, as well as manually control devices from Apple HomeKit-compatible gadgets.

Keywords. MODELING, APPLE HOMEKIT, AUTOMATION, SMART HOME.

ЗМІСТ

ВСТУП

Автоматизація є важливим напрямком розвитку підприємств та практики управління бізнес-процесами. Вона забезпечує ефективність роботи завдяки програмно-апаратним системам, які збільшують швидкість виконання та зменшують виробничі затрати. Інтернет речей в свою чергу забезпечує величезні апаратні та програмні можливості для реалізації проектів незалежно від сфери застосування. В своїй суті він представляє системи для отримання, обробки та видачі даних, що вже є автоматизованим процесом. Говорячи про інтернет речей потрібно розглядати усі аспекти – пристрої, моделі взаємодії, канали взаємодії, засоби обробки та зберігання даних.

У плані обробки та зберігання даних стрімко розвиваються хмарні сервіси, які забезпечують реалізацію широкого набору послуг та сервісів, що скорочує витрати на розробку, впровадження та підтримку систем. Великим плюсом є оплата лише за використані ресурси, що наряду з «розумними» вузлами, які значну частину роботи обробляють власноруч і така система забезпечує ефективність, стабільність та масштабованість рішень.

Тому актуальним є дослідження можливостей Smart Grid, зокрема інтернету речей, в поєднанні з хмарними сервісами [9], зокрема, як ефективного рішення для автоматизації процесів в сфері торгівлі, так і в якості уніфікованого рішення для автоматизації процесів в будь-якій галузі або “розумному будинку” [6].

Незважаючи на вже існуючі програмні рішення, необхідно постійно вдосконалювати технології, використовуючи сучасні розробки та методи рішення певних проблем. Управління житлом на відстані, виконання команди через деякий період часу, підвищення безпеки та швидкодія програми, кросплатформність — всі вище перелічені пункти вимагають використання новітніх приладів та алгоритмів [20]. Розвиток нових технологій в цифровій, інформаційній сферах та сфері іт, дає змогу розробити більш комплексний продукт, який дозволяє повністю автоматизувати приміщення або житловий чи промисловий комплекс.

Одними з найважливіших завдань дипломної роботи є розробка і створення програмного додатку для збору та обробки даних з пристроїв, приєднаних до додатку. В даній роботі розглядаються основні шляхи вирішення даної проблеми, зважаючи на всі переваги та недоліки реалізації поставленої задачі. Програмне забезпечення складається з окремих мікросервісів [26], що надає можливість у майбутньому впровадити дане рішення для інших систем. Кожен мікросервіс представляє собою окреме технічне рішення, що у сукупності з іншими, утворює єдину систему моніторингу даних показників певної споруди [8].

Для реалізації даного програмного забезпечення було обрано GraphQL Apollo та мову програмування JavaScript.

У першому розділі пояснювальної записки сформульовано мету й описано постановку задачі.

У другому розділі наводяться приклади схожих реалізованих систем.

У третьому розділі обґрунтовано вибір програмних засобів.

У четвертому розділі описується програмна реалізація.

У п’ятому розділі подано методику роботи користувача з програмною системою.

У шостому розділі описано розробку стартап проекту.

1. ЗАДАЧА МОДЕЛЮВАННЯ ПРОЦЕСІВ МІЖАГЕНТНОЇ ВЗАЄМОДІЇ В МЕРЕЖАХ SMART GRID

Метою роботи є створення веб-середовища для моделювання процесів міжагентної взаємодії в умовах “розумного будинку”.

Для досягнення поставленої мети необхідно:

- проаналізувати роботу існуючих систем;
- розширити теоретичні знання у відповідній тематиці;
- розробити засоби взаємодії з пристроями;
- розробити засоби керування сервером;
- розробити графічний інтерфейс;
- розробити засоби взаємодії з пристроями, які підтримують Apple HomeKit;
- здійснити програмну реалізацію системи;
- побудувати макет діючої системи.

Програмний додаток має включати наступний функціонал:

- можливість додавання доступних типів датчиків у вже існуючу систему;
- візуалізація отриманих даних;
- створівати автоматизовані сценарії;
- створювати заплановані сценарії;
- додавати дії до сценаріїв;
- ручне керування пристроями;
- керування пристроями з пристроїв, які підтримують Apple HomeKit;
- втоматична синхронизація з Apple HomeKit.

Кожному пристрою в даній системі відповідає запис у базі даних з унікальною адресою. В додатку доступні три типи керування виконавчими пристроями:

- автоматичним сценарієм;
- запланованим сценарієм;
- вручну.

Комунікація між додатком та присроями відбувається через протокол зв’язку MQTT.

Основними користувачами програмної системи є:

- люди, які прагнуть використовувати дану систему в себе вдома;
- підприємці, які прагнуть використовувати систему у офісних приміщеннях.

Також, система повинна бути масштабованою. Це означає, що у майбутньому це дає можливість додати нові приміщення до системи без необхідності купувати нову. Також масштабованість дає можливість користувачеві самому визначати необхідні сенсори і функції системи при цьому не позбавляючи можливість додати їх у майбутньому. З цих та інших причин дуже важливо, щоб в «Розумному» будинку можна було додавати нові функції і пристрої (вертикальне розширення) або нові приміщення (горизонтальне розширення). Виробники часто підтримують обідва типи розширення за допомогою розробки системи на одній мові мережі, 19 наприклад IP (Internet Protocol), а також можливістю бездротового дооснащення продуктами, які можуть взаємодіяти за допомогою існуючої домашньої мережі або провідних пристроїв.

2. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

У кожній сучасній споруді (будинку) в тій чи й іншій мірі функціонує велика кількість обладнання, що забезпечує побут, комфорт, затишок, зв'язок і безпеку, що допомагає відпочити і створює повноцінне робоче середовище. Зручність управління цими системами, їх інтеграція один з одним, можливість злагоджено працювати разом, збільшуючи тим самим функціональність кожної з них окремо - все це і дає можливість назвати такий будинок - Розумним домом.

У відсутності людини Розумний будинок буде підтримувати оптимальним чином постійний мікроклімат, зберігаючи тим самим затишок, кімнатні рослини і меблі. Вона вимкне не потрібен світло або навпаки буде створювати видимість вашої присутності, включаючи і вимикаючи освітлення в тій або іншій кімнаті час від часу. Розумний будинок дозволить Вам спокійно і безтурботно відпочивати.

Розумний будинок буде постійно стежити за всіма інженерними системами в будинку і не допустить спалаху або вибуху пов'язаного з витоком газу або зіпсування меблів через витік води. Також не залишиться непоміченим проникнення в будинок сторонньої особи.

На даний момент існує багато систем, які автоматизовують певні процеси у приміщенні [7]. Серед них є як одиничні системи, які складаються з пристроїв одного типу, так і комплексні, які вміщують в собі пристрої різних типів. Далі пропонується опис програмних рішень для автоматизації приміщень.

2.1. Програмна система Xiaomi Smart Home

Smart House — це програма, розроблена компанією Xiaomi, для централізованого курування будинком. Дана програма спроможна миттєво

підключаються до різних датчиків, замків, вимикачів та інших сумісних пристроїв. Програма спроможна співпрацювати з пристроями від різних виробників.

Дана система спроможна:

- співпрацювати з пристроями, та синхронізувати їх роботу;
- контролювати якість освітлення, температуру тощо;
- запобігати взлому;
- керувати пристроями на відстані;
- Задавати скаладні сценарії.
- Основними недоліками є:
- не всі пристрої співпрацюють з даною системою;
- дані з пристроїв відсилаються на сервер компанії;
- контроль на відстані працює лише за наявності інтернет зв’язку.

Інтерфейс системи зручний і дозволяє легко управляти як одним так і групою пристроїв одночасно. Комунікація між пристроями відбувається через бездротовий протокол ZigBee, між користувачем і хабом відбувається через Wi-Fi з’єднання (рисунок 2.1).

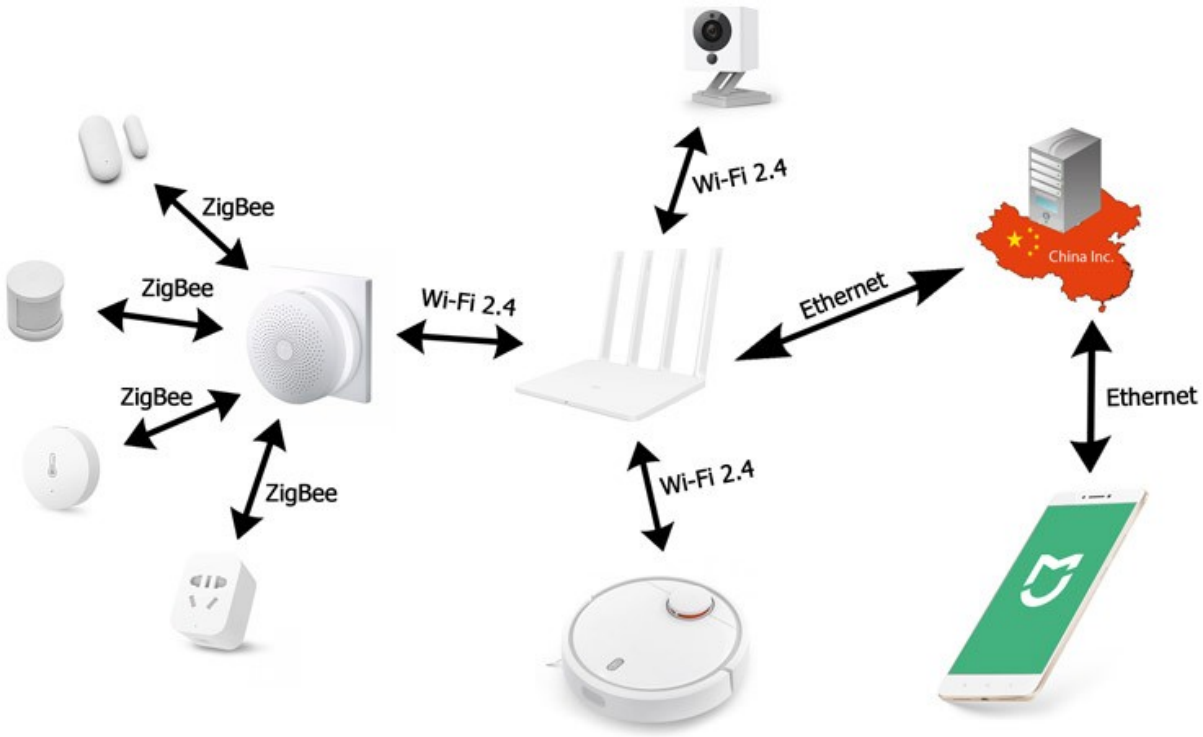


Рисунок 2.1 — Комунікація між пристроями та користувачем в системі Smart House

2.2. Програмна система Apple HomeKit

Apple HomeKit — це програмне забезпечення, закладене в IOS, яке об’єднує різні додатки для керування будинком та утворює єдиний загальний центр керування. Основною первагою даної системи є надійний захист інтелектуального будинку від вторгнення та несанкціонованого доступу сторонніх осіб.

Програма дозволяє управляти пристроями трьома основними способами: режим “Home”, режим “Room” та режим “Automation”. Режим “Home” представляє собою список пристроїв, якими користувач найчастіше користується. За допомогою режиму “Room” управляються пристрої, які оточують будинок. Режим “Automation” дозволяє автоматизувати інтелектуальну поведінку пристроїв та систем у будинку, наприклад, увімкнути сигналізацію чи вимкнути

світло при виході з приміщення. Користувач може створювати окремі режими, в яких декілька пристроїв будуть працювати як єдина автоматизована система.

Нові пристрої можна додати, скануючи код із шести цифр, проте користувачу слід окремо зберігати дані коди на випадок коли знову прийдеться додавати пристрій в систему. Також клієнт може створювати нові коди для деяких пристроїв, але це не легка процедура, яка потребує сторонніх додатків.

Основними недоліками є:

- дана програма розповсюджена лише для пристроїв фірми Apple;
- складно під’єднати нові пристрої до домашньої системи.

2.3. Програмна система Works with Nest

Works with Nest — це система, розроблена компанією Nest, яка дозволяє централізовано управляти приладами у будинку. Даний продукт легко співпрацює з іншими платформами, що дає змогу користувачу без залучення додаткового програмного забезпечення, підключитися до різних наборів пов’язаних технологій. Даний продукт дозволяє легко керувати різними побутовими системами в будинку такими як: система освітлення, система вентиляції, опалення, відеоспостереження тощо.

Переваги:

- Гнучність налаштувань;
- Інтегровність з сторонніми сервісами.

Недоліки:

- Дороговизна пристроїв;
- Обмежений функціонал при керуванні через стороннє ПО.

2.4. Програмна система Node-RED

Node-RED — інструмент для візуального програмування потоком даних, розроблений працівниками компанії IBM для поєднання різноманітних пристроїв, API та онлайн-сервісів як складових частин Інтернету речей. Node-RED дає змогу працювати з браузерним редактором потоків даних як окремими вузлами з різним функціоналом, що уможливлюють створення JavaScript-функцій. Причому можна використовувати як базові вузли, якими одразу забезпечений Node-RED, так і встановлювати вузли з додатковим функціоналом з репозиторію npm [18] або ж навіть створити свій власний вузол з унікальним функціоналом. Програми або ж їхні частини, розроблені за допомогою Node-RED, можуть бути збережені та поширені для вільного використання. Саме середовище побудовано на основі Node.js. Потоки, створені за допомогою Node-RED, зберігаються у вигляді JSON.

Дана система спроможна:

- співпрацювати з пристроями, та синхронізувати їх роботу;
- контролювати пристроями за правилами користувача;
- створювати правила у графічному інтерфейсі.
- Основними недоліками є:
- складність налаштування системи системи.

2.5. Програмна система Google Cloud Platform

Хмарна платформа надає можливості, які можна використовувати в своїх інтересах для щоденних операцій (рисунок 2.2) [21]:

- Google Cloud Monitoring надає панелі моніторингу та оповіщення для хмарних додатків. На пристроях Linux можна встановити Cloud Monitoring agent, який є Stackdriver на основі системи агента. Крім того, можна використовувати API Monitoring Cloud з потрібники метриками.

– Google Cloud Logging збирає і зберігає журнали, таким чином, можна переглядати, шукати, фільтрувати і експортувати інформацію. Використання Cloud Logging може заощадити багато часу і зусиль у порівнянні з побудовою рішення на замовлення.

– Google Cloud audit logs охоплює адміністрацію та доступ до даних діяльності, пов'язаної з Cloud Platform, зберігаючи їх у незмінних журналах, які можуть бути використані для аудиту.

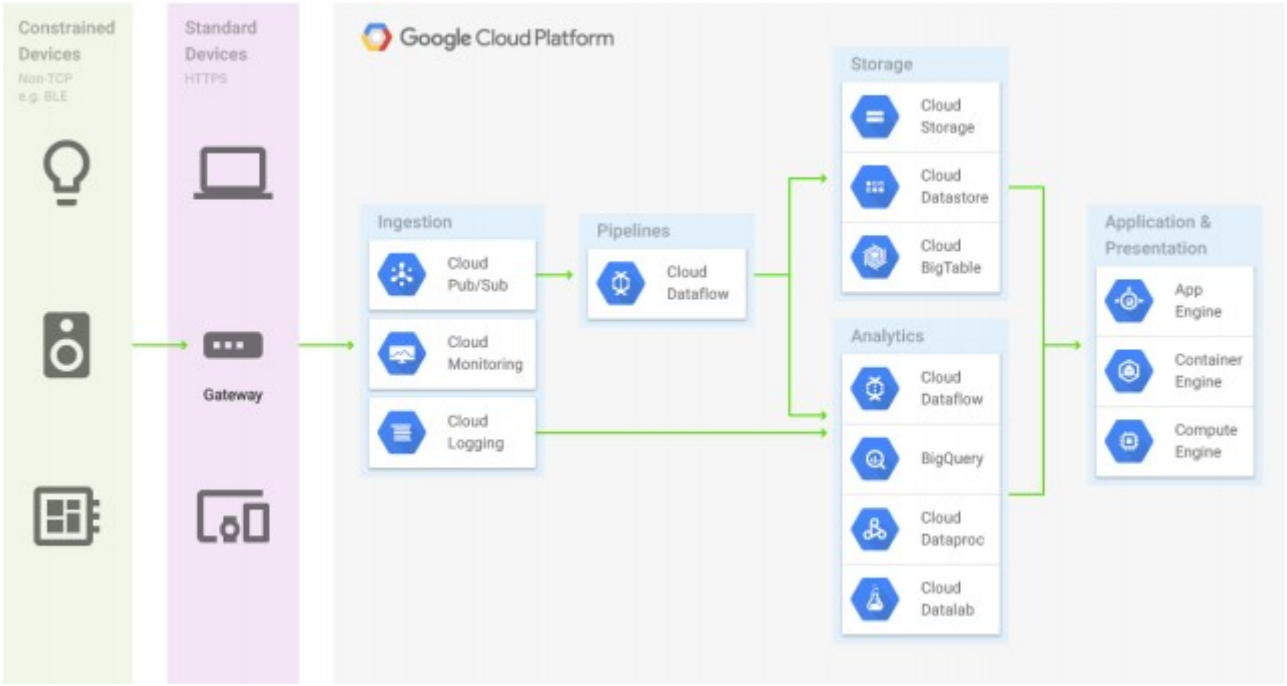


Рисунок 2.2 — Різні етапи управління даними IoT в Google Cloud Platform

В Google Cloud Pub / Sub створюючи теми для потоків або каналів, можна включити різні компоненти застосування, щоб підписатися на певні потоки даних без необхідності побудови індивідуальних абонентських каналів на кожному пристрої. Cloud Pub / Sub також спочатку підключається до інших сервісів Cloud Platform, допомагаючи підключити прийом даних, шлюзів і систем зберігання даних.

Cloud Pub / Sub може діяти як амортизатор і зрівнювач швидкості для обох вхідних потоків даних і зміни архітектури додатків. Багато пристроїв мають

обмежені можливості для зберігання і надсилання даних телеметрії. Платформа може використовуватися для обробки даних шипи, які можуть 28 виникнути, коли багато пристроїв реагують на події в фізичному світі, і буферизувати ці шипи, щоб допомогти ізолювати їх від додатків моніторингу даних.

Дані з фізичного світу приходить в різних формах і розмірах. Хмарна платформа пропонує широкий вибір рішень для зберігання даних з неструктурованих згустків даних, таких як зображення або відео потоків, структурованого зберігання показників пристроїв або транзакцій.

Стан пристрою в загальному випадку може бути змодельований як набір пар ключ-значення. Деякі пристроїможуть бути підключені безпосередньо до 29 апаратних засобів. Інший стан пристрою може існувати на рівні додатків. Часто цінно, навіть необхідно, для іншого програмного забезпечення, такого як мобільний додаток або веб-сайт [16], прочитати або змінити останній стан цього пристрою. З огляду на те, що IOT пристрої можуть провести деякий час в малопотужний режим сну і може існувати на особливо ненадійних мережах, часто буває корисно відобразити деякі стани для пристрою з хмарою. Таким чином, дані стану можуть бути зроблені доступними навіть тоді, коли самі пристрої тимчасово відсутній.

Дана система має наступні переваги:

- Інструменти аналітики;
- Гнучкість налаштувань.

Недоліи:

- Складність налаштування;
- Висока ціна програмного забезпечення;
- Не велика кільсть сумісниї пристроїв.

Порівняння рішень наведено у таблиці 3.1.

Таблиця 3.1 – Порівняння аналогів систем “розумного будинку”.

Виробн	Власна	Xiaomi	Apple	Works	Node-	Google

ик Параметр	розробка	Smart House	Home Kit	With Nest	Red	cloud platform
Вартість	Низька	Середня	Висока	Висока	Низька	Висока
Функціонал	Базовий	Базовий	Базовий	Розширений	Майже необмеженний	Розширений
Взаємодія компонентів	хаб	хаб	пристрій	хмарний додаток	хаб	хмара
Масштабованість	+	+	+	-	+	+
Складність налаштування	Не складно	Не складно	Не складно	Не складно	Складно	Складно

3. ЗАСОБИ ТА МЕТОДИ РЕАЛІЗАЦІЇ ВЕБ-СЕРЕДОВИЩА ДЛЯ МОДЕЛЮВАННЯ ПРОЦЕСІВ МІЖАГЕНТНОЇ ВЗАЄМОДІЇ

Одним із найважливіших завдань при розробці є вибір таких засобів, які б полегшили роботу програміста, надавши всі необхідні інструменти для реалізації поставленого завдання.

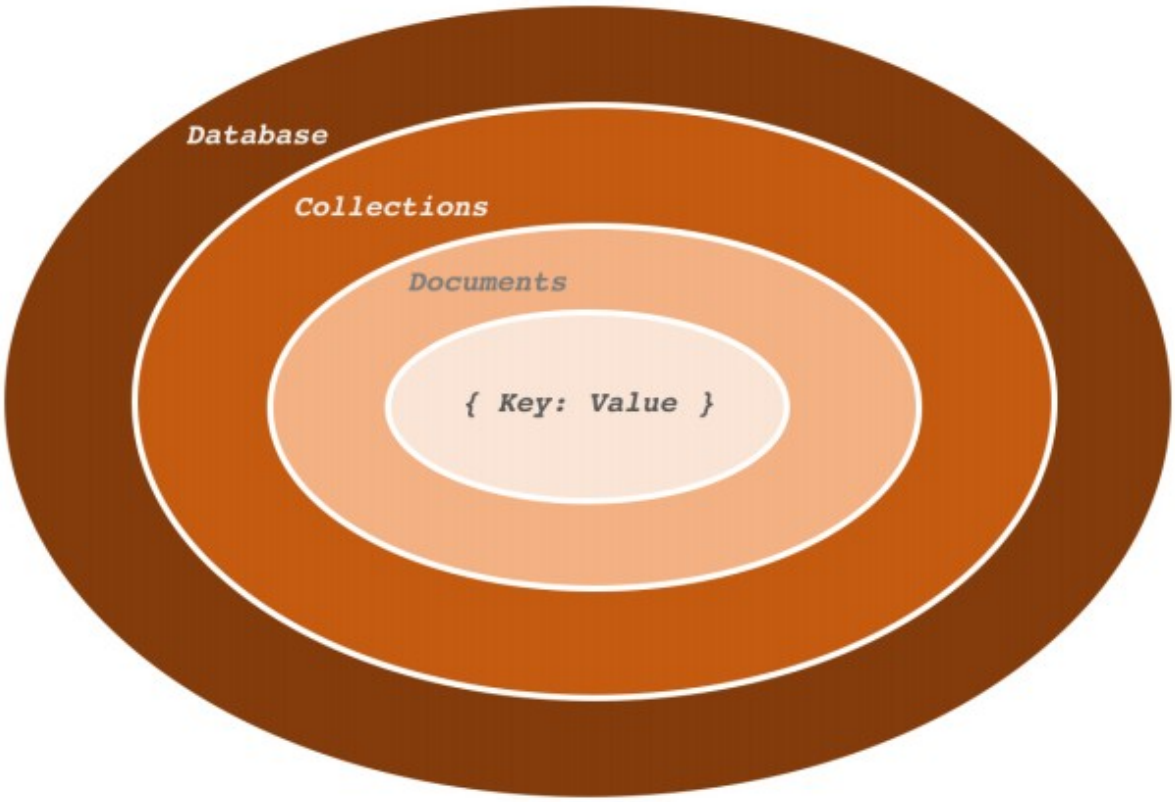
При створенні додатку для контролю пристроїв було обрано наступні технології: JavaScript, MQTT, Node.js, GraphQL, Docker, MongoDb та ReactJs.

3.1. База даних

База даних MongoDB реалізує новий підхід до побудови баз даних, де відсутні таблиці, схеми, запити SQL, зовнішні ключі та багато інших особливостей, які властиві об’єктно-реляційним базам даних. MongoDB класифікується як NoSQL база даних. На відміну від реляційних СКБД, MongoDB пропонує документо- орієнтовану модель даних, що дає значний приріст у швидкості роботи масштабованості.

MongoDB має наступну структуру (рисунок 3.1):

- база даних, кожна з яких є контейнером для інших сутностей;
- база даних має «колекції». Колекція настільки схожа на традиційну «таблицю», що можна сміливо вважати їх одним і тим же;
- колекції складаються з «документів». Документ можна розглядати як «рядок».



Р

исунок 3.1 – Структура СУБД MongoDB

Архітектура MongoDB має дві особливості:

– Не існує такого поняття як «транзакція». Атомарність гарантується лише на рівні цілого документа. Це означає, що не може статися часткового оновлення документа.

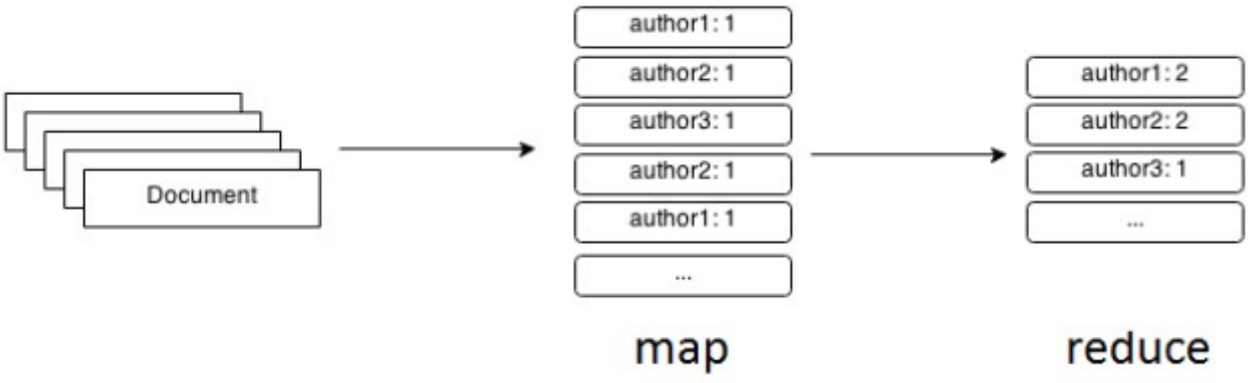
– Відсутнє поняття «ізоляції». Будь-які дані, які зчитуються одним клієнтом, можуть змінюватися іншим клієнтом паралельно.

СКБД MongoDB керує наборами JSON-подібних документів, які зберігаються у двійковому вигляді у форматі BSON (Binary JSON). Структура BSON дозволяє набагато швидше працювати з даними, оскільки швидше виконується пошук та обробка. Слід відмітити, що BSON у порівнянні з JSON займає більше місця, але з іншої сторони, цей недолік повністю перекривається швидкістю роботи. На відміну від таблиць у реляційних базах даних, MongoDB має колекції. І якщо реляційні бази даних обмежені тим, що таблиці можуть зберігати однотипні, жорстко структуровані об’єкти, то колекція може містити

об’єкти, які мають різну структуру і абсолютно різний набір властивостей. Якщо реляційні бази даних зберігають рядки, то MongoDB зберігає документи. Це дозволяє зберігати складну по структурі інформацію. Документ можна уявити як сховище ключів та значень. Ключ являє собою просту мітку, з яким асоціюється певний набір даних. Кожному ключу співвідноситься певне значення. І якщо у реляційних базах даних є чітко окреслена структура, де є поля, які обов’язково повинні мати якесь значення (NULL, якщо значення порожнє), то в MongoDB, якщо ключу не співставлене значення, то такий ключ просто опускається і не використовується в документі. MongoDB підтримує ad hoc запити, які можуть повертати конкретні поля документів і користувацько JavaScript функції. Крім того, є підтримка пошуку по регулярним виразам regex, а також можна налаштувати запит на повернення випадкового набору значень.

MongoDB дозволяє працювати за набором реплік. Набір реплік містить дві або більше копії даних. Всі операції читання та запису проводяться з основною реплікою, а допоміжні підтримують копії даних в актуальному стані. У випадку виникнення збою в основній репліці, таким чином інформація не буде втрачена. До того ж, допоміжні репліки можуть додатково слугувати джерелом для операцій читання з сервера.

MongoDB може працювати з потужним інструментом агрегації даних MapReduce. MapReduce – це модуль розподілених обчислень, який може виконувати паралельні обчислення над величезними об’ємами даних. Робота MapReduce складається з двох кроків (рисунку 3.2). На кроці map проходить попередня обробка вхідних даних. Для цього головний вузол (master node) отримує вхідні дані, розбиває їх на частини та транслює їх до робочих вузлів (worker node) для попередньої обробки. На кроці reduce відбувається згортання попередньо оброблених даних. Master node отримує відповіді від робочих вузлів і формує результат на їх основі.



Рис

унок 3.2 – Приклад застосування парадигми MapReduce

Вся система MongoDB може являти собою не одну базу даних, яка знаходиться на одному фізичному сервері. Функціональність MongoDB дозволяє розташувати декілька баз даних на декількох фізичних серверах і мати можливість вільного обміну даними між цими базами та збереження цілісності.

MongoDB ідеально підходить для роботи з Node.js. JavaScript може використовуватися прямо у запитах та функціях агрегації та відправлятися в базу для виконання. Завдяки своїй архітектурі, MongoDB надає широкі можливості для масштабування, швидкої роботи та доступу до даних.

Node.js має спеціальний модуль для комфортної роботи з MongoDB – Mongoose.js. Mongoose – це спеціальна бібліотека ODM (англ. «Object Document Mapper» - об’єктно-документований відображувач) [10]. Вона дозволяє визначати об’єкти зі строго типізованою схемою, яка відповідає документу MongoDB.

Mongoose надає великий набір функціональних можливостей для створення та роботи зі схемами. На даний момент Mongoose містить вісім типів даних схеми SchemaTypes, зокрема String, Number, Date, Buffer, Boolean, Mixed, ObjectId, Array.

Для кожного типу є можливість:

- задати значення за замовчуванням;
- задати власну функцію перевірки даних;
- вказати обов’язкове заповнення поля;

– задати get-функцію для маніпуляції з даними до їх повернення у вигляді об’єкта;

– задати get-функцію для маніпуляції з даними до їх збереження у базу даних;

– визначення регулярного виразу, який дасть можливість у процесі перевірки даних обмежити дозволені для збереження варіанти даних;

– визначення переліку, який дозволяє встановити список допустимих рядків.

Тип даних mixed дає можливість зберігати дані будь-якого типу, що надає певної гнучкості для розробника. Але не слід цим зловживати, оскільки перевірка даних такого типу, а також відслідковування змін сутності таких даних обмежені.

Mongoose надає декілька функцій для пошуку даних певної моделі. Якщо потрібно написати складний запит, mongoose пропонує aggregate API (рисунок 3.3).

```
/**
 * Finds last values for sensor
 * @param {String} sensorId id of sensor
 * @returns {Promise<Value>} promise with found last values
 */
async function findLastValuesBySensorId(sensorId) {
  return await this.aggregate()
    .match({sensorId})
    .sort({'created_at': 'desc'})
    .group({_id: {sensorId: '$sensorId', type: '$type'}, doc: {$first: '$$ROOT'}})
    .exec();
}
```

Рисунок 3.3 – Приклад пошуку даних у Mongo

3.2. Засоби контейнеризації

Оскільки кількість мікросервісів може бути доволі великою та їхня технологічна природа може дуже відрізнятися, то команда розробників стикається з проблемою управління різноманітними середовищами для запуску сервісів. Для

простоти створення різних необхідних програмних середовищ використовуються контейнери для інкапсуляції мікросервісів.

Найпопулярнішим інструментом для контейнеризації є Docker [29]. Контейнеризація як альтернатива віртуалізації, завжди мала потенціал змінити шлях розгортання додатків. Docker як реалізація інструменту контейнеризації, часто порівнюється з віртуальними машинами. Віртуальні машини були створені з метою оптимізації використання обчислювальних ресурсів. На одному сервері можливо запустити декілька віртуальних машин та розгорнути окремі додатки на кожній з віртуальних машин. За цією моделлю, кожна з віртуальних машин надає стабільне програмне середовище для кожного додатку. Але при масштабуванні додатку ми отримуємо значні проблеми з продуктивністю, оскільки віртуальна машина споживає багато системних ресурсів.

Оскільки мікросервіси є відносно маленькими програмами, які необхідно розміщати в окремому програмному середовищі, то створення цілої віртуальної машини не є ефективним підходом. З Docker можливо зменшити витрати на 51 продуктивність та розгортати велику кількість сервісів на єдиному сервері, тому що Docker-контейнер вимагає набагато менше ресурсів.

Основні переваги використання Docker:

- Швидкий час запуску. Контейнер запускається в межах декількох секунд, оскільки він є процесом операційної системи, в той час запуск окремої віртуальної машини займе декілька хвилин.
- Швидше розгортання. Для контейнеру немає потреби кожен раз налаштовувати середовище, а лише необхідно завантажити відповідний образ.
- Просте управління та масштабування контейнерів.
- Краще використання обчислювальних ресурсів.
- Підтримка великої кількості різних операційних систем.

3.3. З’єднання пристроїв

З’єднання пристрїв з додатком здійснюється по протоколу MQTT. MQTT (англ. Message Queue Telemetry Transport) — це простий відкритий протокол, який був розроблений спеціально для застосування в IoT і обміну даними між пристроями [12]. MQTT-мережа складається з MQTT-брокера (broker), який виступає посередником у взаємодії MQTT-агентів – видавців (publisher) і підписників (subscriber). Видавці публікують інформацію, призначену для підписників (рисунок 3.4).

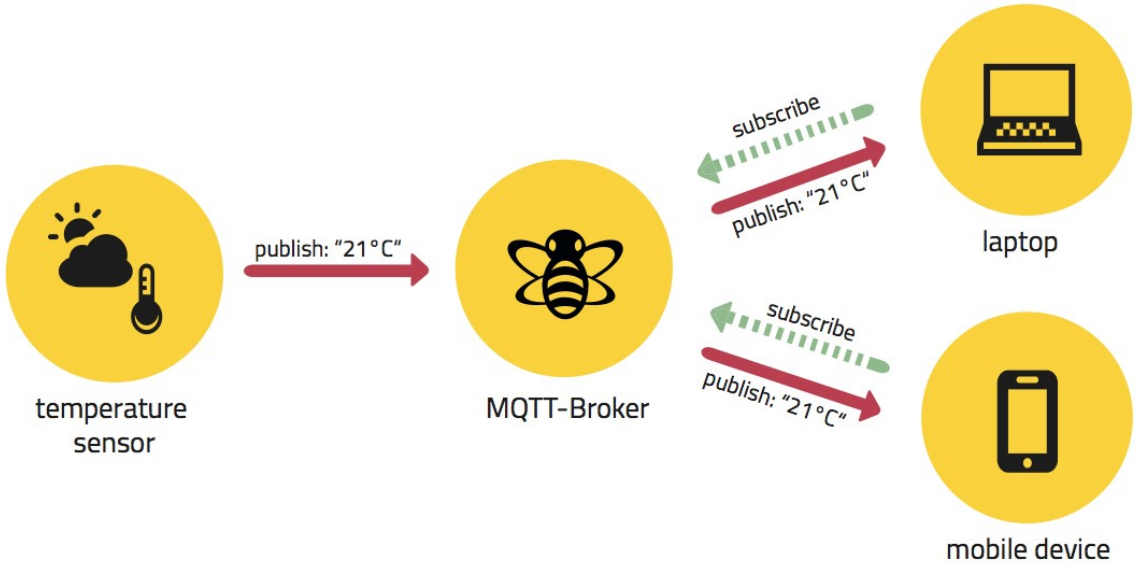


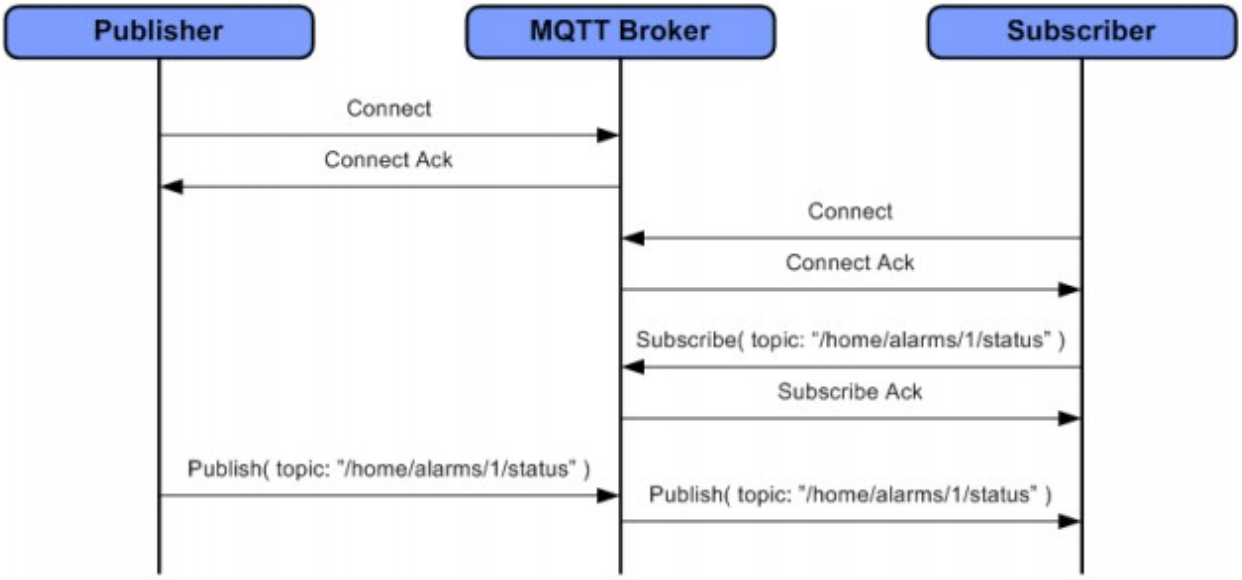
Рисунок 3.4 —Взаємодія між пристроями

MQTT працює за моделлю «видавець - підписник», та використовує мінімальну кількість методів. Вони служать для вказівки дій, які необхідно виконувати. Ці дії зводяться до комунікації з брокером і до роботи з темами і повідомленнями. Агенти підключаються до брокера, а потім або публікують теми і повідомлення в них, або підписуються на теми і отримують повідомлення, в цих темах опубліковані. Завершивши роботу, агент відключається від брокера. Ось як виглядають методи MQTT:

- Connect - встановити з'єднання з брокером;
- Відключити - розірвати з'єднання з брокером;

- Опублікувати - опублікувати тему на брокера;
- Підписка - підписатися на тему на брокера;
- Відмовитися від підписки - відписатися від теми на брокера.

Діаграма послідовності обміну повідомленнями між підписником і видавцем з використанням MQTT-брокера наведена на рисунку 3.5.



Ри

сунок 3.5 – Діаграма посдідовності обміну повідомленнями в мережі MQTT

MQTT підтримує вказівку рівня якості (QoS Обслуговування). А саме, існують три таких рівні:

- QoS 0. Цей рівень задіє стратегію «максимум одноразова доставка повідомлень». Приймач повідомлення не підтверджує їх отримання, відправник, відповідно, передає повідомлення лише раз, не роблячи спроб по їх повторної передачі. Це - метод «відправив і забув».
- QoS 1. Тут застосовується підхід «мінімум одноразова доставка повідомлень». Гарантується, що приймач отримає повідомлення хоча б один раз. При цьому підписник може отримати одне й те саме повідомлення кілька разів. А відправник буде робити повторні спроби відправки до тих пір, поки не отримає підтвердження в успішній доставку повідомлення.

− QoS 2. Цьому рівню якості обслуговування відповідає найповільніша процедура доставки повідомлень, але при цьому він - найнадійніший. Його основна особливість - реалізація стратегії «одноразова доставка повідомлень». При його використанні застосовується чотириступінчаста процедура підтвердження доставки повідомлень.

Вибір конкретного рівня якості обслуговування залежить від особливостей переданих даних і від того, наскільки важливо, щоб вони були доставлені.

У термінах MQTT транспортуються дані і метаінформація, що формує «канали» транспорту, представлені MQTT UTF-8 рядками. Рядки метаінформації формуються з фрагментів, що мають назву топіків (topic) або тем, передбачені спеціальні символи і правила форматування, що вводять ієрархію топіків і можливість «підписки» них.

Ієрархія формується з'єднанням тем за допомогою символу «/».

3.4. Розробка серверної частини

Серверна частина була написана на Node.js з використанням RabbitMQ [25], GraphQL. Node.js — платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript [2]. Перш за все, Node.js відрізняється від класичного JavaScript тим, що виконуваний код виконується на стороні сервера (backend), а не на стороні браузера [11]. Для інтерпретації коду Node.js використовує движок V8, який в даний час застосовується в Google Chrome. Крім того, всі механізми обробки запитів та інших операцій введення / виводу (I / O) побудовані на подіях. Це означає, що в Node.js немає ніякого способу, щоб заблокувати працючий в даний момент потік.

Кожна операція в Node.js виконується асинхронно. Це є величезною перевагою, особливо якщо код повинен бути побудований на операціях введення-виведення: читання дисків, підключення до бази даних, веб-сервіси і т.д. На відміну від IIS або Apache, Node.js не використовують многопоточную модель. У

Node.js є тільки один робочий потік, який обслуговує всі запити користувачів і відповідають ресурси. Існує POSIX басейн асинхронних потоків Node.js, який містить безліч асинхронних потоків для операцій введення-виведення. Цикл між користувачем і інтерфейсом POSIX дає можливість передавати дані туди і назад, а самі асинхронні операції здійснюються в POSIX. Дана модель отримала назву event-driven non-blocking IO model. Продуктивність в такій системі набагато вище, ніж, якщо використовувалася многопоточная модель (multi-threaded blocking model).

Для здійснення налаштувань серверу користувачем, було обрано GraphQL. GraphQL [24] - це синтаксис, який описує як запросити дані і, в основному, використовується клієнтською стороною для загрузки даних з серверу. Дана технологія має три основні характеристики:

- дозволяє клієнтській стороні точно вказати, які дані потрібні;
- полегшує агрегацію даних з декількох джерел;
- використовує систему типів для опису даних.

GraphQL – асинхронний, тому користувач може повноцінно користуватись сторінкою, поки сервер ще обробляє запит.

GraphQL навідмінну від REST [17] набагато комфортніший в використанні, тому що API [15] створене за допомогою GraphQL дозволяє створювати endpoint, який працює зі складними запитами та надає даним таку форму, яку потребує клієнтська сторона. Наприклад, якщо клієнту потрібні різні ресурси, то потрібно виконувати декілька запитів, щоб отримати усі потрібні дані(REST), але навіщо, якщо можна створити один запит на потрібні ресурси, а GraphQL їх збере і поверне у потрібному вигляді.

Приклад GraphQL запиту:

```
query getLocalenamespace($key: ID!) {
    locales(namespace: $key) {
        key
        ru
```

```
            en
        }
    }
```

Традиційні серверні системи зазвичай будуються як моноліти – логічно окремі виконувані програми. І цей підхід є природнім: уся логіка обробки запитів працює в одному процесі, що дозволяє використовувати наявні інструменти мови програмування для поділу програми на класи [1], функції та простори імен. Моноліт можна масштабувати горизонтально, запустивши багато екземплярів поза балансувальником навантаження.

Монолітне ПЗ є досить успішним, але має свої недоліки. Цикли зміни моноліту зв’язані один з одним, тобто зміна невеликої частини системи вимагає повторного збирання (компіляції) і розгортання. З плином часу стає складно підтримувати гарну модульну структуру, утримуючи зміни, що стосуються конкретного модуля, лише всередині нього. Масштабувати доводиться весь моноліт замість окремих частин, які потребують більшої кількості ресурсів.

Мікросервіси ж можна розгортати та масштабувати незалежно один від одного. Вони також забезпечують чіткі межі між модулями, навіть дозволяючи реалізовувати окремі підсистеми на різних мовах програмування. Таке розмежування допомагає керувати складністю кодової бази, оскільки кожен модуль матиме публічний API, який міститиме лише потрібну функціональність, а все інше буде інкапсульовано та не мати значення для розробки інших сервісів, які залежать від цього.

Основним недоліком мікросервісної архітектури є збільшення складності обробки помилок. На відміну від моноліту, кожен виклик сервісу може закінчитися невдачею. Тому потрібно розробляти механізми моніторингу стану сервісів, перевіряти різноманітні метрики їх функціонування, а також автоматизовувати відновлення мікросервісу в разі його відмови.

Позитивним моментом є те, що відмови в мікросервісній архітектурі є великою мірою ізольованими. Якщо задоволення запиту вимагає виклику багатьох сервісів і частина з них недоступна, можливо надати менш повну відповідь (graceful degradation).

Складності також можуть бути викликані забезпеченням послідовності розгортання залежних мікросервісів, а також необхідністю керування транзакціями, які взаємодіють із декількома підсистемами.

3.5. Розробка інтерфейсу користувача

React - відкрита JavaScript бібліотека для створення інтерфейсів, яка покликана вирішувати проблеми часткового оновлення вмісту вебсторінки, з якими часто стикаються в розробці односторінкових додатків. Розробляється Facebook, Instagram і співтовариством індивідуальних розробників.

React дозволяє розробникам створювати великі веб-додатки, які використовують дані, мінливі з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабується. React обробляє в програмах, тільки інтерфейс [3].

Базові поняття React:

Елементи - це JavaScript об'єкти, які представляють собою HTML - елементи. Їх не існує в браузері, вони описують DOM-елементи, такі як div, h1 або button. Компоненти - це елементи React, написані розробником. Зазвичай це частини призначеного для користувача інтерфейсу, які містять свою структуру і функціональність. Наприклад, такі як NavBar, LikeButton, або ImageUploader.

Властивості - опції компонента. Вони надаються в якості аргументів компонента і виглядають так само, як атрибути HTML.

Стан - це спеціальний об'єкт всередині компонента. він зберігає дані, які можуть змінятся з часом.

Композиція - комбінування менших компонентів при формуванні більшого.

Особливості:

− Одностороння передача даних. Властивості передаються в рендерер компонента, як властивості html тега. Компонент не може безпосередньо змінювати властивості. Однак компонент може мати внутрішній стан. Це об'єкт this.state, доступний всередині самого компонента.

− Віртуальний DOM. React підтримує віртуальний DOM (Document Object Model). Це дозволяє бібліотеці визначити, які частини DOM змінилися в порівнянні із збереженою версією віртуального DOM, і таким чином визначити, як найефективніше оновити DOM браузера.

− Компоненти React зазвичай написані на JSX - розширення синтаксису JavaScript, що нагадує XML, яке дозволяє використовувати синтаксис HTML тегів для рендеринга компонентів Код написаний на JSX компілюється в виклики методів бібліотеки React.

React на відміну від більш великих MVC-фреймворків вирішує щодо вузьке завдання: рендеринг інтерфейсу [5]. Він, надзвичайно швидкий, оскільки використовує віртуальний DOM і оновлює тільки змінені частини сторінки. А також надає прості засоби для створення адаптивно-гібридних або ізоморфних веб-додатків.

React розроблений навколо концепції багаторазових компонентів. Ви визначаєте невеликі компоненти, і поєднуєте їх, щоб сформувати більші компоненти. Всі компоненти, маленькі чи великі, можуть використовуватися повторно, навіть у різних проектах.

Принцип MVC у веб-програмуванні (Model - View - Controller, Модель - Подання (Вид) - Контролер) - одна з найбільш вдалих ідей на сьогоднішній день [14]. Принцип MVC інтуїтивно зрозумілий на перший погляд, але не дуже простий при поглибленні. Спочатку розглянемо, для чого він призначений.

Принцип MVC, дозволяє розділити реалізацію логіки докладання, зовнішній вигляд (графічний інтерфейс, GUI) і взаємодія з користувачем.

Це призводить до більш структурованого коду, дозволяє працювати над проектом більш спеціалізованим людям, спрощує підтримку коду, робить його більш логічним і зрозумілим. Зміна в одному з компонентів мінімально впливає на інші. Можна до однієї моделі підключати різні види, різні контролери.

Розглянемо докладніше компоненти.

Model (Модель) - містить так звану "Бізнес-логіку" - обробку і верифікацію даних, звернення до баз даних, представляє внутрішній устрій системи. Модель не повинна безпосередньо взаємодіяти з користувачем.

View (Вид, Подання) описує зовнішній вигляд програми.

Controller (Контролер) - сполучна ланка між моделлю і видом, отримує дані від користувача, передає їх моделі, отримує оброблений результат і передає його в уявлення.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ТА МАКЕТУ СИСТЕМИ

Програмна реалізація додатку представляє собою сім взаємодіючих компонентів:

- клієнтський веб-інтерфейс;
- сервіс, що забезпечує взаємодію клієнта та сервера;
- сервіс, що забезпечує взаємодію пристроїв та сервера.
- сервіс, що забезпечує взаємодію пристроїв на iOS та сервера.
- сервіс, що забезпечує виконання сценаріїв за розкладом.
- сервіс, що забезпечує виконання сценаріїв автоматично.

Основні компоненти системи наведені на рисунок 4.1

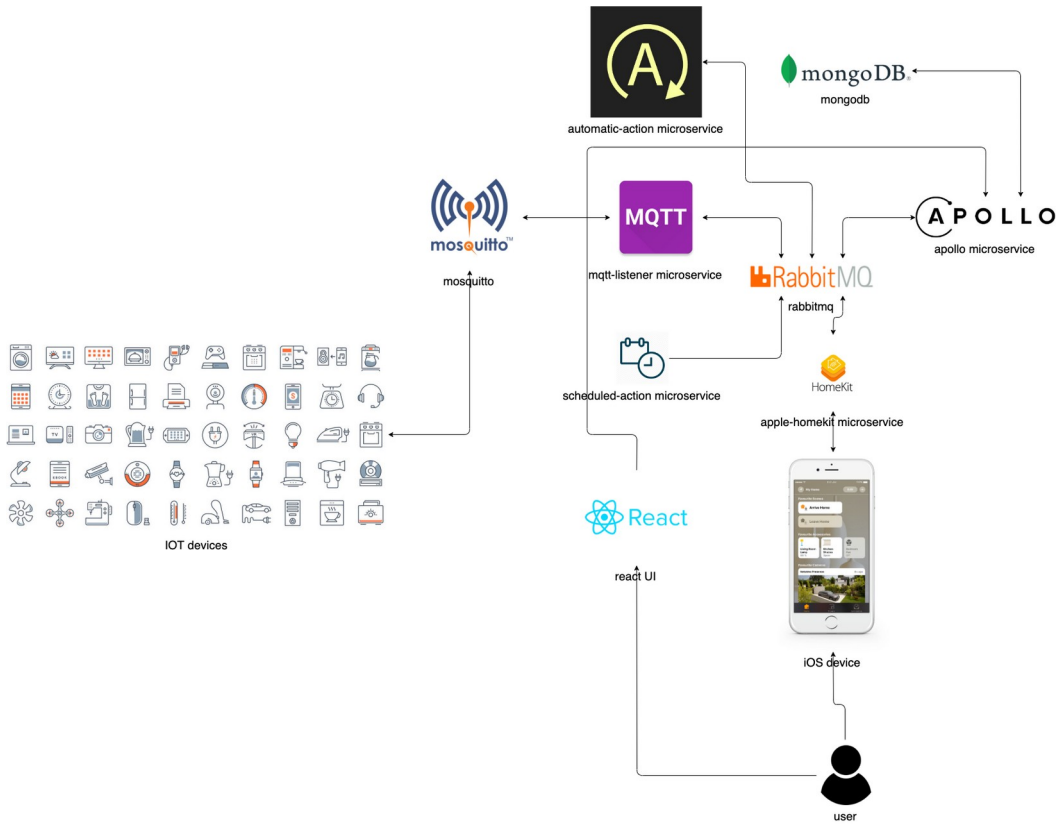


Рисунок 4.1 – Загальна архітектура додатку

Далі буде детально розглянуто кожен із складових компонентів системи.

4.1. Функції системи

На діаграмі прецендентів (рисунок 4.2) представлено основні функцію додатку— створення автоматизованих та запланованих сценаріїв.

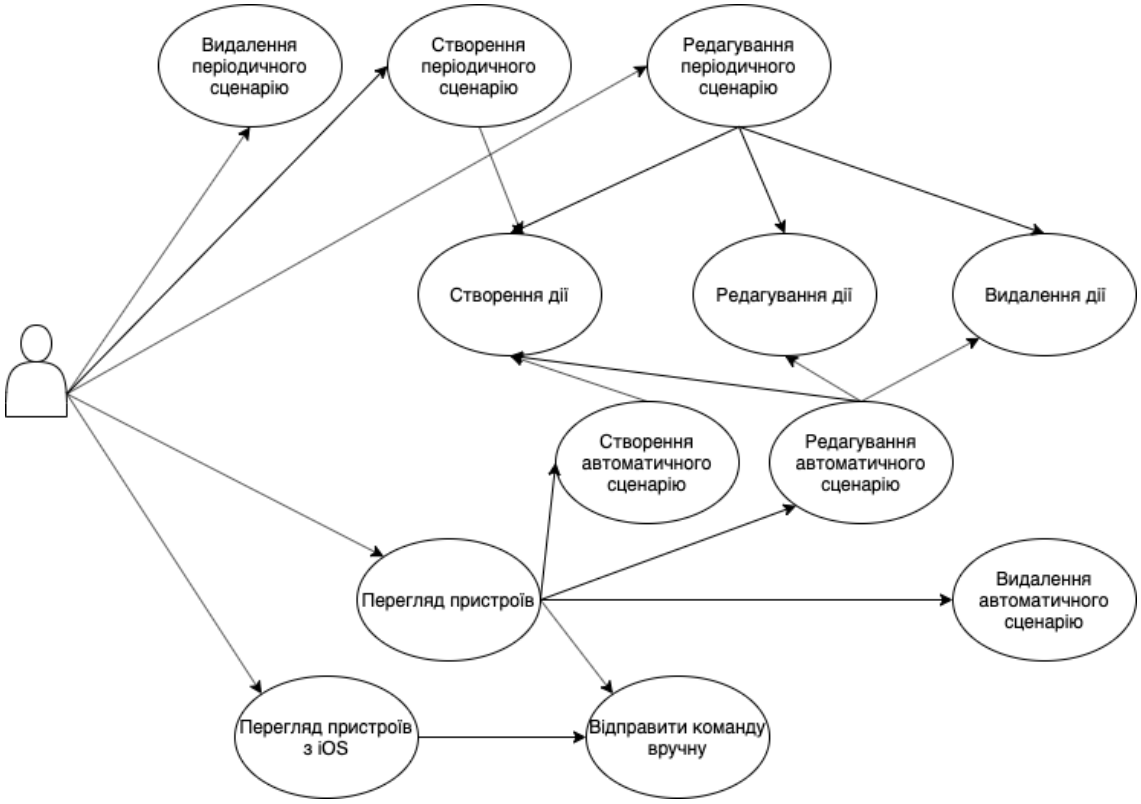


Рисунок 4.2 — Діаграма прецедентів

4.2. Структура бази даних

Для автоматизації дій пристроїв було розлобно сутності (рисунок 4.3):

- Sensor – зберігає з’єднанні пристрої;
  - name – назва сенсору;
  - type – тип сенсору;

- description – опис сенсору;
- mqttStatusTopic –mqtt адреса для зчитування даних;
- mqttSetTopic – mqtt адреса для відправлення даних.

❖ Value – містить отримані дані від сенсорів;

- sensor –id сенсору;
- createdAt – дата отримання значення;
- value – отримане значення;
- deleted – флаг, який вказує, чи видалений запис.

❖ ScheduledAction – зберігає заплановані сценарії;

- name – назва сценарію;
- schedule – розклад сценарію у форматі cron;
- enabled – стан сценарію (активний чи ні);
- deleted – флаг, який вказує, чи видалений запис;
- actions – дії, які будуть виконанні.

❖ AutomaticAction – зберігає автоматичні сценарії;

- name – назва автоматично сценарію;
- sensor – id сенсору;
- valueToCompare – значення з яким порівнювати;
- condition – умова порівняння;
- enabled – стан сценарію (активний чи ні);
- deleted – флаг, який вказує, чи видалений запис;
- actions – дії, які будуть виконанні.

❖ Action – зберігає дії до виконання.

- sensor – id сенсору;
- automaticAction – id автоматичного сценарію;
- scheduledAction – id сценарію за розкладом;
- valueToChangeOn – значення, яке буде встановлено.
- deleted – флаг, який вказує, чи видалений запис.

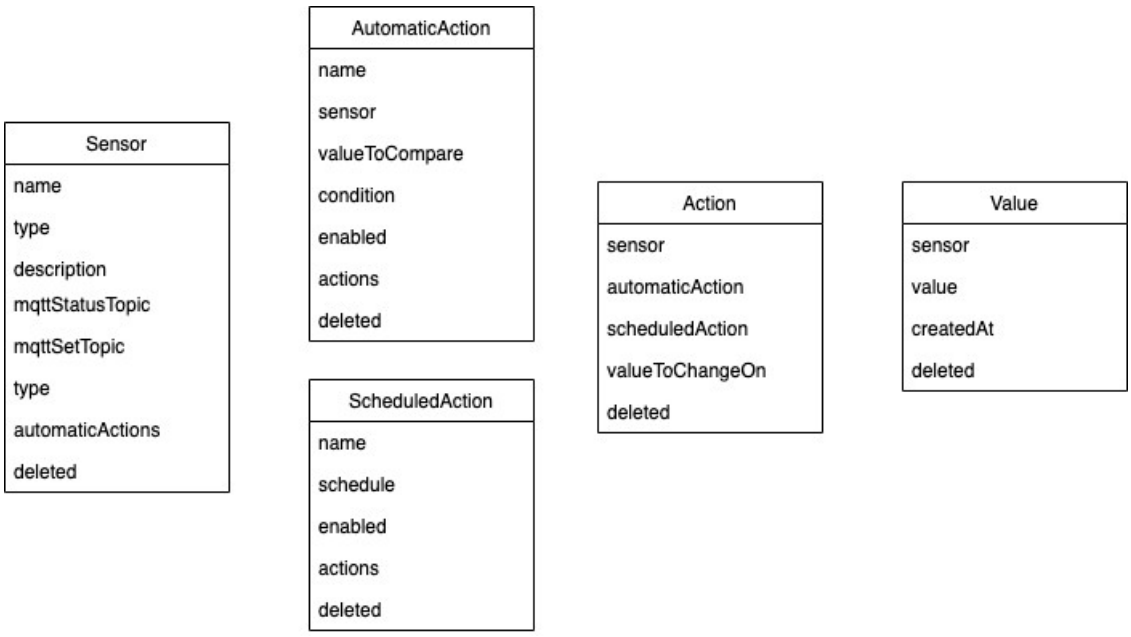


Рисунок 4.3 — Структура бази даних

4.3. Структура модулю взаємодії пристрою та сервера

Користувач може використовувати пристрої, які представляють інформацію лише у числовому вигляді. З'єднання пристрою з додатком здійснюється за допомогою MQTT-брокера. Було використано MQTT-брокер mosquitto.

Mosquitto - це брокер повідомлень з відкритим сирцевим кодом (під ліцензією EPL/EDL), що реалізовує протокол MQTT версій 3.1 та 3.1.1. Mosquitto є доволі легкою системою, та внаслідок цього може використовуватись на багатьох видах пристроїв: від простих одноплатних комп'ютерів до повноцінних серверів.

Було розроблено мікросервіс mqtt-listener для взаємодії з пристроями, який слухає rpc чергу повідомлень від інших мікросервісів та mqtt повідомлення від пристроїв.

Для того, щоб відіслати новий стан до пристроїв – необхідно встановити mqtt адресу через веб-інтерфейс, яку прослуховує пристрій.

При з’єднанні пристрою з додатком, відбувається створення запису про пристрій у базі даних та оновлення списку сенсорів у Apple HomeKit. Якщо сенсор не відповідає жодному з сконфігурованих, пристрій буде одразу відключено.

При отриманні даних від пристроїв, відбувається перевірка на автоматичний сценарій.

4.4. Структура модулю взаємодії користувача та сервера

В основі розроблювальної системи полягає клієнт-серверна архітектура. Клієнт робить запит до сервера на виконання певних дій або представлення деякої інформації, сервер отримує запит, виконує його та відправляє відповідь.

Внаслідок архітектурних рішень було створено мікросервіс apollo, який надає користувачу можливість самостійно налаштовувати будь-які параметри системи. Завдяки GraphQL та Apollo [23] було створено лише одну функцію, яка відповідає на запити користувача для всіх типів, та ще одну функцію для реалізації підписки на всі типи.

Функціонал підписки [32] на колекції дає можливість отримувати дані у реальному часі. Також підписка на колекції підтримує фільтрування. Приклад підписки з фільтрування на рисунку 4.4.

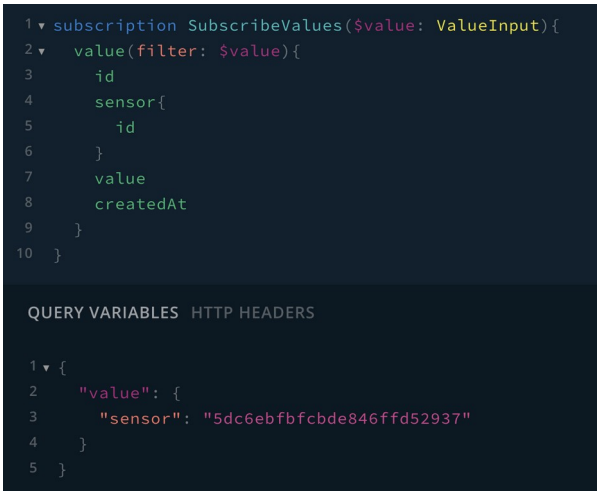


Рисунок 4.4 — приклад підписки на нові дані з пристроїв

Apollo надає можливість вказувати для кожної колекції поля, які треба обробляти (рисунок 4.5) — це дає можливість в одному запиті до сервера отримати всі необхідні вкладені колекції.

```
const resolvers = {
  Subscription: {
    sensor: subscriber('sensor'),
    automaticAction: subscriber('automaticAction'),
    scheduledAction: subscriber('scheduledAction'),
    action: subscriber('action'),
    value: subscriber('value'),
  },
  Query: {
    sensor: resolver,
    sensors: resolver,
    automaticAction: resolver,
    automaticActions: resolver,
    scheduledAction: resolver,
    scheduledActions: resolver,
    value: resolver,
    values: resolver,
    appleHomeKit: resolver,
  },
  Mutation: {
    sensor: resolver,
    automaticAction: resolver,
    scheduledAction: resolver,
    value: resolver,
  },
  AutomaticAction: {
    actions: resolver,
    sensor: resolver,
  },
  ScheduledAction: {      Dmytro Shvaika, 3 months
    actions: resolver,
  },
  Action: {
    sensor: resolver,
    automaticAction: resolver,
    scheduledAction: resolver,
  },
  Sensor: {
    automaticActions: resolver,
  },
  Value: {
    sensor: resolver,
  }
}
```

Рисунок 4.5 — налаштування apollo

Також мікросервіс Apollo прослуховує rpc чергу повідомлень від інших мікросервісів. Результатом запитів до мікросервісу є зчитування, створення, редагування або видалення даних з бази даних.

4.5. Мікросервіс автоматичних дій

Для виконання автоматичних дій було створено мікросервіс automatic-action. При отриманні даних від пристроїв, мікросервіс mqtt-listener відправляє отримані дані до мікросервісів a pollo та automatic-action.

При отриманні даних від мікросервісу mqtt-listener, мікросервіс automatic-action виконує rpc запит до мікросервісу apollo, щоб отримати список автоматичних сценаріїв разом з діями — які будуть застосовані. Після отримання сценаріїв — відбувається їх фільтрування за умовою зазначеною користувачем. Зі сценаріїв, які задовільняють умовам — відправляються через rpc до мікросервісу mqtt-listener щоб відправити нові значення на пристрої.

4.6. Мікросервіс періодичних дій

Для сценаріїв, які необхідно запускати за розкладом — було створено мікросервіс scheduled-action.

При ініціалізації мікросервісу — робиться запит до мікросервісу apollo, щоб отримати список сценаріїв разом з діями, які будуть засвтосовані.

Якщо відбувається будь-яка зміна у періодичному сценарії або зв’язаної з ним дії — apollo надсилає rpc запит до мікросервісу scheduled-action щоб той перезавантажив періодичні сценарії.

Коли настає час виконання сценарію — scheduled-action надсилає нові значення для пристроїв до mqtt-listener.

4.7. Мікросервіс Apple HomeKit

Розроблений додаток надає можливість керувати пристроями не лише з веб-інтерфейсу, але й з пристроїв на Apple, які підтримують HomeKit.

Мікросервіс apple-homekit розроблений на базі npm модулю HAP-NodeJS [22], який реалізує протокол взаємодії з iOS пристроями HomeKit Accessory Server [27].

Для з’єднання пристрою на iOS з додатком — необхідно сканувати qr код (рисунок 4.6). Всі пристрої, які були під’єднанні до додатку автоматично стунуть доступні у HomeKit.

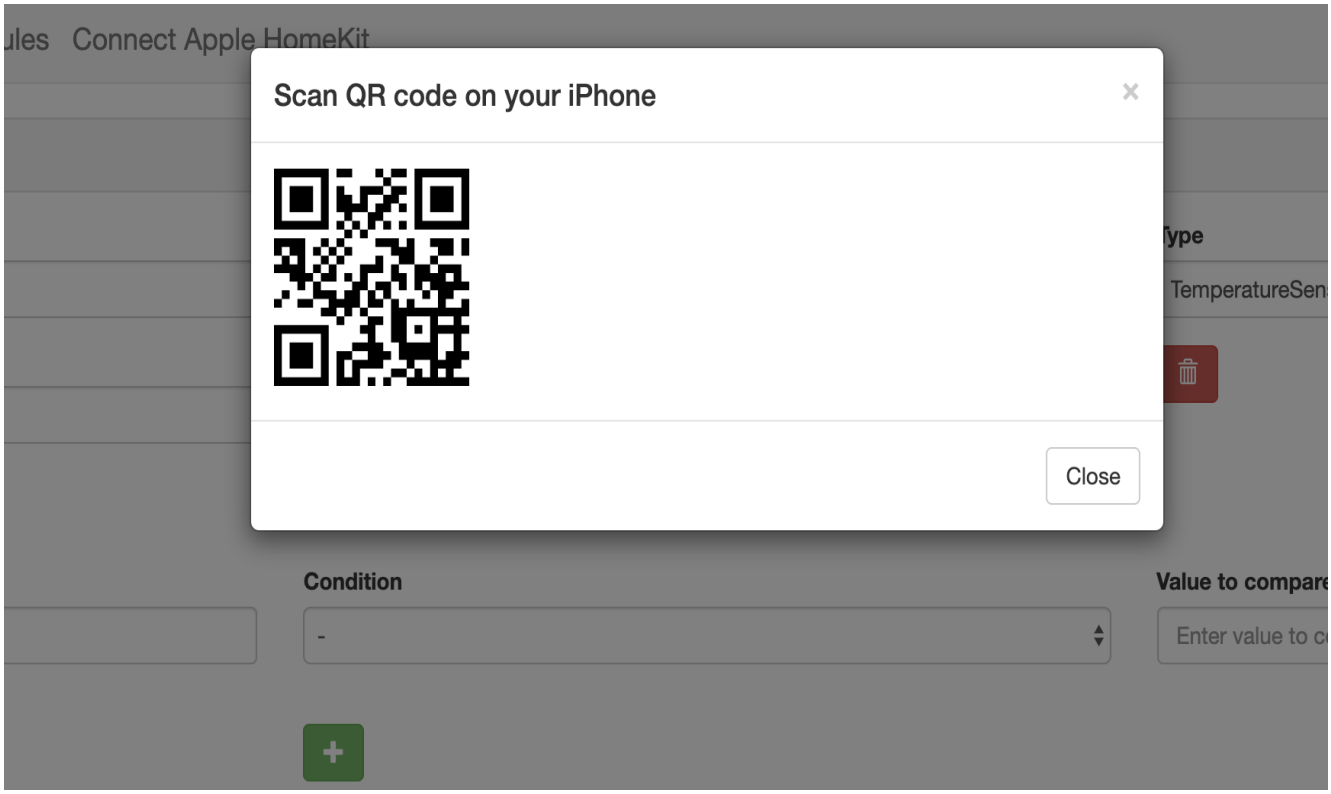


Рисунок 4.6 — з’єднання iOS пристрою з додатком

Якщо змінюється будь-яке поле, створюєтьсяу або видаляється запис у колекціїї пристроїв, мікросервіс apollo надсилає rpc запит до мікросервісу apple-homekit для оновлення списку пристроїв.

На данний час екосистема Apple дозволяє створювати автоматичні сценарії та сценарії за розкладом лише якщо у вас є AppleTv, але додаток вирішує цю проблему. Єдиний недолік у цьому випадку — можливість переглядати, створювати та видаляти сценарії є лише з веб-інтерфейсу.

Якщо користувач керує пристроями за допомогою Apple HomeKit — то нові значення для пристроїв відправляються з мікросервісу apple-homekit до mqtt-listener.

4.8. Технології для розробки інтерфейсу

Для навігації у веб-інтерфейсі замість окремиах сторінок, було використано модуль React Router (рисунок 4.7). React Router — це невеликий модуль для навігації у веб-інтерфейсі.

```
const App = () => (
  <ApolloProvider client={client}>
    <Router>
      <div>
        <Route exact path="/" component={SensorsList} />
        <Route path="/scheduled_actions" component={SchedulesActionsList} />
      </div>
    </Router>
  </ApolloProvider>
)
```

Рисунок 4.7 — Навігація у веб-інтерфейсі за допомогою ReactRouter

У результаті такого підходу, розробка інтерфейсу значно прискорюється, на приклад не сервері значно менша кількість адес запитів для відображеня інтерфейсу.

Було створено наступні компоненти:

- SensorsList — компонент для відображення списку сенсорів;
- Sensor — компонент для сенсору;
- AutomaticActionsList — компонент для відображення списку автоматичних сценаріїв;
- AutomaticAction — компонент для відображення автоматичного сценарію;
- ScheduledActionsList — компонент для відображення списку запланованих сценаріїв;
- ScheduledAction — компонент для відображення запланованого сценарію;

- ActionsList — компонент для відображення списку дій у сценарії;
- Action — компонент для відображення дії у сценарії;
- Header — компонент для відображення заголовку.

Кожен компонент представляє собою самостійний, незалежний блок, що дозволяє використовувати його будь де, на приклад на рисунку 4.8 блок, що відображає один пристрій. Він також використовує компонент «Sensor» для відображення налаштувань конкретного сенсору.

```
return (
  <div>
    <Header />
    {
      data.sensors.map(sensor => (
        <Sensor key={sensor.id} sensor={sensor} />
      ))
    }
  </div>
)
```

Рисунок 4.8 — Компонент SensorsList

Як можна бачити з прикладу коду компонента SensorsList на рисунку 4.8, було використано jsx синтакс. На данний час браузери не підтирумують jsx, тому було застосовано транспілятор babel. Він перекладає нові можливості JavaScript зрозумілі браузерам, використовуючим старі версії JavaSript.

Для того щоб скомпілювати інтерфейс, потрібно у директорії з проектом виконати команду “npm run watch”. Побудований файл буде знаходитись у директорії “./public/javascripts/built/”. Після цього веб-інтерфейс готовий до роботи.

4.9. Діючий макет системи

Для комплексного тестування системи було розроблено два пристрої. За основу було взято мікроконтроллер ESP8266 виробництва Espressif. Програму для них було розроблено в середовищі Arduino IDE [28].

Пристрій з сенсором складається з мікроконтроллеру ESP8266 та сенсору температури і вологості DHT22, підключеного за схемою на рисунку 4.9.

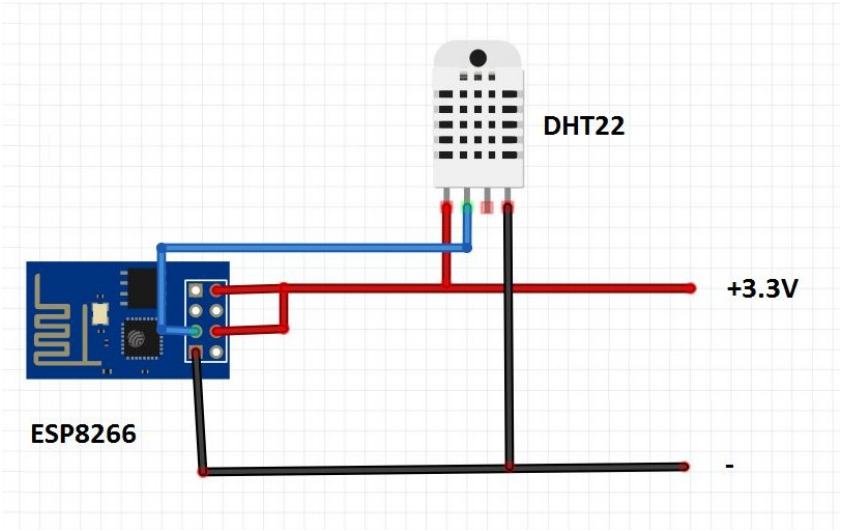


Рисунок 4.9 — Схема підключення пристрою з сенсором

Для зв’язку з сервером було використано бібліотеку з відкритим кодом PubSubClient [19]. Ця бібліотека надає інструменти для зв’язку пристрою з MQTT-брокером. Програму для читання данних з сенсору (температури та вологості) та відправлення їх на сервер через MQTT для мікроконтроллера зображено на рисунку 4.10.

```
int tellstate = 0;
if ( (millis() - tellstate) > INTERVAL ) {
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  if (isnan(t) || isnan(h)) {
    return;
  }
  client.publish(writeTopicTemp, String(t).c_str());
  client.publish(writeTopicHum, String(h).c_str());
  tellstate = millis();
 }
}
```

Рисунок 4.10 — Приклад програми для відправлення даних з пристрою по MQTT

Виконуючий пристрій складається з мікронтроллеру ESP8266 та реле, підключеного за схемою на рисунку 4.11.

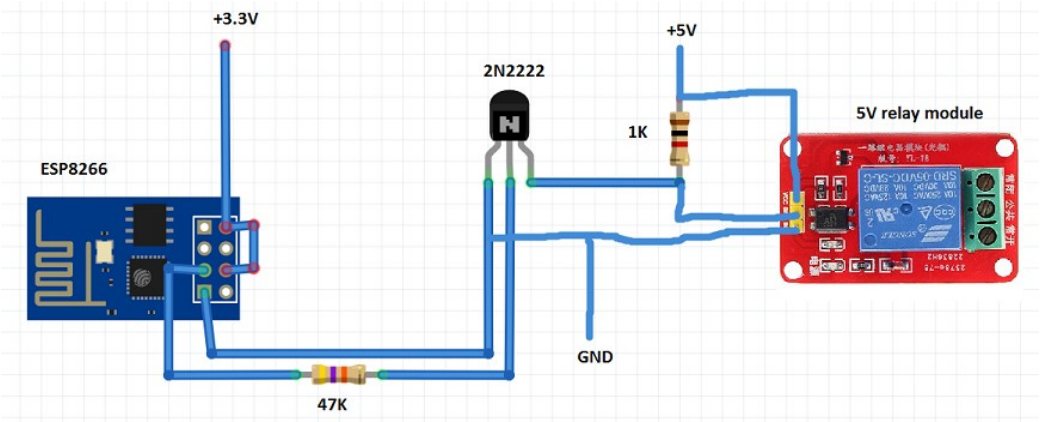


Рисунок 4.11 — Схема підключення пристрою з реле

Програму контролю реле для пристрою показано на рисунку 4.12.

```
int RelayPin = 5;    // RELAY connected to digital pin 5
const int INTERVAL = 10000;
const char *ssid =  "                ";
const char *pass =  "             .";
const char* mqtt_server = "192.168.1.35";
const char* listenRelayTopic = "cmnd/relayNode/relay1/RELAY/";
const char* writeRealayTopic = "stat/relayNode/relay1/RELAY/";
WiFiClient wclient;
PubSubClient client(wclient);

void callback(char* topic, byte* payload, unsigned int length) {
  payload[length] = '\0';
  if ( String((char*)topic) == String((char*)listenRelayTopic) ) {
    if (String((char *)payload) == "1" ) {
      digitalWrite(RelayPin, HIGH);   // turn the RELAY on
      client.publish(writeRealayTopic, "1");
    } else if ( String((char *)payload) == "0" ) {
      digitalWrite(RelayPin, LOW);    // turn the RELAY off
      client.publish(writeRealayTopic, "0");
    }
  }
}

void connect_to_MQTT() {
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
  if (client.connect("relayNode")) {client.subscribe(listenRelayTopic);}
}

void setup() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
  connect_to_MQTT();
  pinMode(RelayPin, OUTPUT);
}

int tellstate = 0;
void loop() {
  client.loop();
  if (! client.connected()) {
    connect_to_MQTT();
    delay(5000);
  }
  if ( (millis() - tellstate) > INTERVAL ) {
    if ( digitalRead(RelayPin) ) {client.publish(writeRealayTopic, "1");} else {client.publish(writeRealayTopic, "0");}
    tellstate = millis();
  }
}
```

Рисунок 4.12 — Приклад програми для пристрою з реле

Як можна бачити з коду програми для пристрою з реле (рисунок 4.12) та інших виконавчих пристроїв, на відміну від пристрою з сенсором — пристрій не тільки відправляє дані на певну адресу, але й прослуховує іншу mqtt адресу для встановлення нового стану.

5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА ІЗ ВЕБ-СЕРЕДОВИЩЕМ ДЛЯ МОДЕЛЮВАННЯ ПРОЦЕСІВ МІЖАГЕНТНОЇ ВЗАЄМОДІЇ В МЕРЕЖАХ SMART GRID

Запуск додатку відбувається за допомогою docker compose [30] — CLI-утиліта для з’єднання контейнерів, для цього потрібно перейти у терміналі у директорію з додатком та ввести команду “docker-compose up”. Далі docker автоматично збудує контейнер для кожного сервісу та запустить їх. Оскількі для мікросервісу apple-homekit потрібен доступ до мережі хосту — то запустити його в оточенні docker не можливо, тому його потрібно запустити вручну. Для цього потрібно перейти у папку з мікросервісом apple-homekit, встановити npm модулі командою “npm i ”, та запустити мікросервіс за допомогою команди “npm run start:preset”.

Кожен сервіс може бути сконфігурований за допомогою файлу .env [31] (рисунок 5.1). В ньому можуть бути сконфігуровані різні параметри, в залежності від мікросервісу, на приклад:

- порт мікросервісу;
- адресу бази даних;
- тип оточення, в якому запущен мікросервіс;
- пін код для з’єднання з iOS;
- порт MQTT-брокеру.

```
1   NODE_ENV=development
2   PORT=51826
3   PIN_CODE=031-45-154
4   BRIDGE_NAME=@MQTT-Bridge
```

Рисунок 5.1 —Конфігураційний файл мікросервісу apple-homekit

Пристрій повинен бути сконфігурований виходячи з конфігурації додатку, на приклад конфігурація пристрою з сенсором на рисунку 5.2.

```
const char* mqtt_server = "192.168.1.35";
const char* writeTopicTemp = "stat/sensorNode/DHT22/TEMP_SENSOR/";
const char* writeTopicHum = "stat/sensorNode/DHT22/HUMIDITY_SENSOR/";
```

Рисунок 5.2 — Конфігурація пристрою з сенсором

Інтерфейс користувача представлений у вигляді списку пристроїв (рисунок 5.3). При розгортанні вклади з назвою пристрою можна побачити загальні налаштування пристрою та налаштування автоматичних дій(рисунок 5.4).

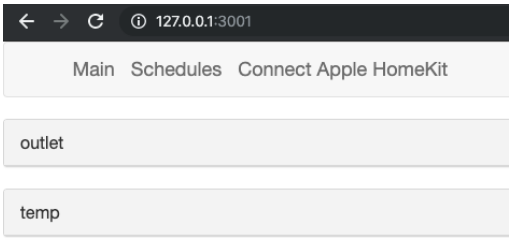


Рисунок 5.3 — Список пристроїв

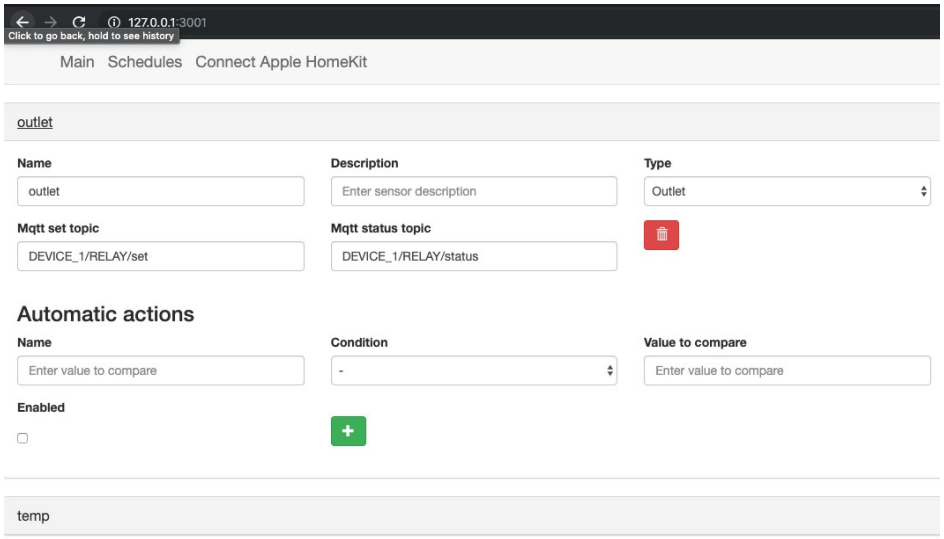


Рисунок 5.4 — Розгорнутий список пристроїв

Кожен пристрій має власні налаштування. На сторінці налаштувань можна побачити загальні налаштування пристрою та форму для додання автоматичного сценарію.

При першому з’єднанні пристрою та додатку створіється запис у базі даних та додаток може отримувати дані від пристрою. Але вказувати новий пристрій у діях до сценаріїв не можливо, оскільки додаток не знає тип пристрою. Для встановлення типу пристрою необхідно в налаштуваннях вибрати відповідний тип (рисунок 5.5).

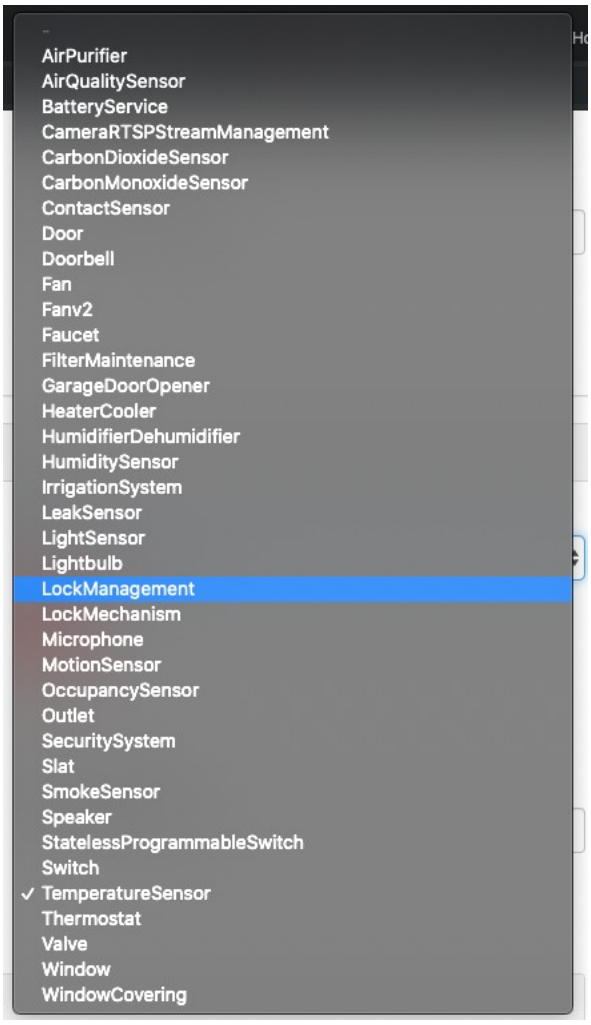


Рисунок 5.5 — Вибір типу пристрою

Якщо обраний пристрій є виконавчим (на приклад розетка), то також необхідно вказати mqtt тему, до якої буде відправлено нове значення на пристрій (рисунок 5.6). Якщо не встановити тему для нових значень, то створені дії будуть ігноруватися.

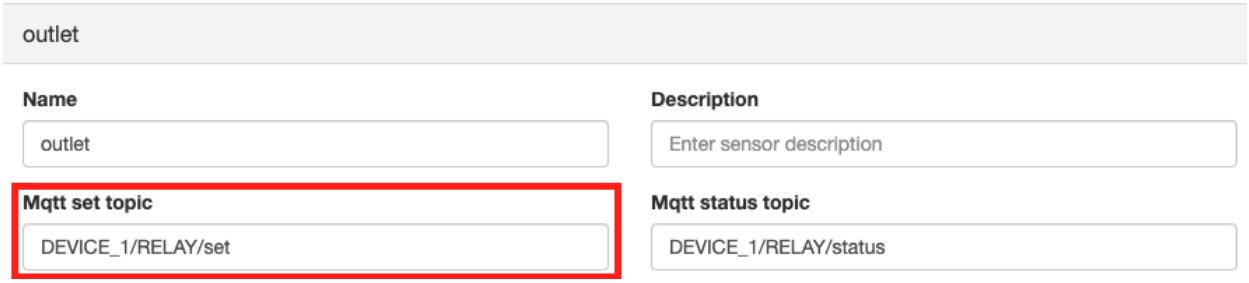


Рисунок 5.6 — Встановлення теми, на яку буде відправлено нове значення

Для створення автоматичного сценарію потрібно ввести назву сценарію, умову порівняння та значення з яким буде порівнюватися отримане значення від пристрою (рисунок 5.7).

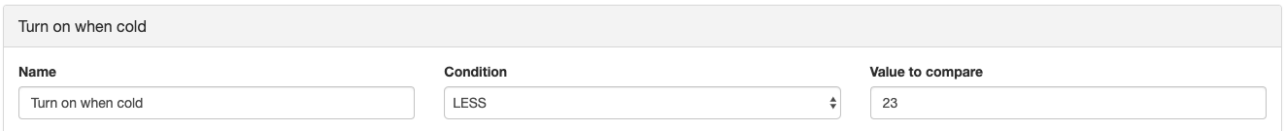


Рисунок 5.7 — Створення автоматичного сценарію

При додаванні дії до сценарію можна вибирати тільки виконавчі пристрої (розетка, лампа, тощо) (рисунок 5.8).

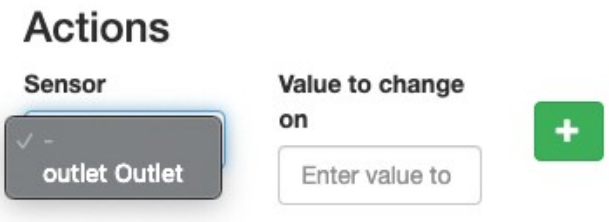


Рисунок 5.8 — Налаштування дій в автоматичному сценарії

Для створення періодичного сценарію треба перейти до відповідної вкладки (рисунок 5.9).

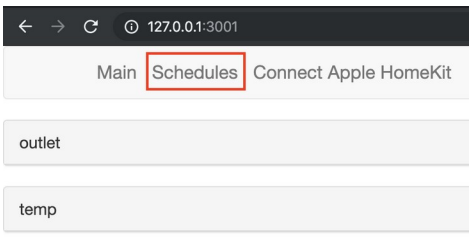


Рисунок 5.9 — Вкладка для налаштування періодичних сценаріїв

При створенні періодичного сценарію необхідно вказати назву сценарію та розклад у форматі крон (рисунок 5.10). Після створення сценарію до нього можна додавати дії (рисунок 5.8). Дії можна створити тільки для виконавчих пристроїв.

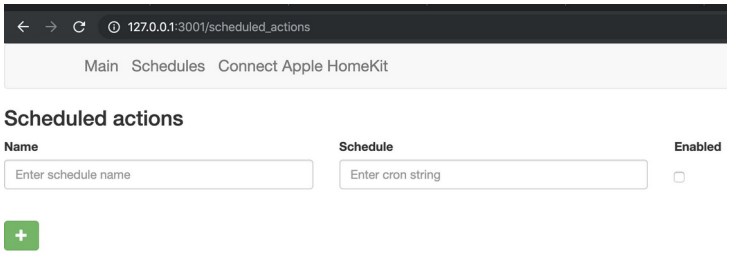


Рисунок 5.10 — Створення періодичного сценарію

Додаток підтримує керування не тільки з веб-інтерфейсу, але й з пристрої Apple, які підтримують HomeKit. Для з’єднання пристрою з додатком необхідно сканувати QR-код (рисунок 5.11).

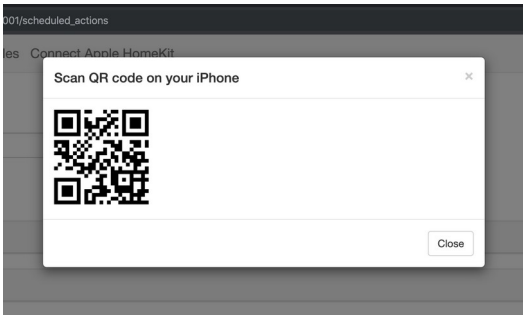


Рисунок 5.11 — QR код для з’єднання пристрою та додатку

Після з’єднання пристрою Apple та додатку — пристрої які були з’єднанні з додатком, та на яких було встановлено тип — автоматично з’являться у переліку

пристроїв у додатку Apple HomeKit, та з’явиться можливість керувати їми вручну (Рисунок 5.12).

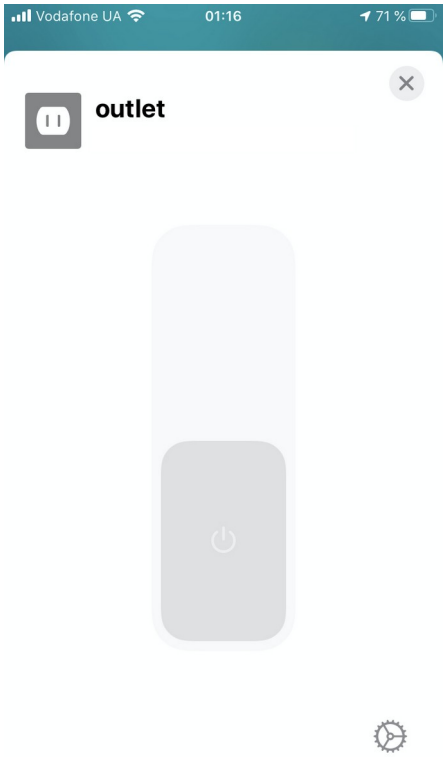


Рисунок 5.12 — Ручне керування розеткою з iPhone

Недоліком керування додатком з пристрою Apple є те, що пристроями можливо керувати лише з одної мережі, та не можливо створювати/редагувати сценарії. Оби дві проблеми може вирішити веб-інтерфейс, або з’єднати додаток з AppleTv. У такому випадку керувати пристроями можливо ззовні.

6. РОЗРОБКА СТАРТАП-ПРОЕКТУ

В даному розділі викладено підхід для розробки рішення, яке реалізує викладене в роботі напрацювання. Провівши аналіз ринку були описані цілі та ідея, сформульовані основні вимоги, визначені сильні та слабкі сторони потенційного комерційного продукту в результаті SWOT-аналізу.

6.1. Опис ідеї проекту

ПЗ відноситься до хмарного сервісу та може бути використане для автоматизації процесів у будинку.

У випадку несумісності приладу та ПЗ – необхідно звертатися до виробника для випуску адаптера для сумісності.

У якості найближчого аналогу прийнято сервіс Apple HomeKit. Сервіс дозволяє керувати приладами на відстані та автоматизувати їх дії.

До недоліків найближчого аналога відноситься невелика кількість сумісних приладів та їх занадто висока вартість.

В основу розробки ПЗ поставлено задачу удосконалення функціоналу вже існуючих на ринку конкурентів, шляхом того, що планується реалізувати підтримку популярних приладів, розробити гнучкий веб-інтерфейс для налаштування автоматизації дій.

Оскільки націлено забезпечити можливість використання широкого кругу пристроїв, що характеризує винахідницький рівень винаходу, то з застосуванням додатку зменшиться вартість та збільшиться вибір пристроїв.

Розглянемо зміст ідеї, можливі напрямки застосування, основні переваги, які зможе отримати користувач представлено у таблиці 6.1.

Таблиця 6.1 – Опис ідеї стартап-проекту.

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Удосконалення програмного	1.Приватні (приватний будинок)	Зменшена вартість розумного будинку, тонке налаштування автоматизованих дій
	2. Комерційні	Застосування при об’єднанні

забезпечення для автоматизації дій у будинку		розумних приладів.

Проблеми:

1. Сумісність з мобільними операційними системами.
2. Інтеграція у вже збудовану інфраструктуру.

Порівняємо стартап ідею з вже існуючими рішеннями (таблиця 6.2).

Таблиця 6.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту.

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів				(слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Apple HomeKit	Xiaomi				
1.	Економічні: Вартість обслуговування; Вартість експлуатації; Вартість утилізації; Вартість витратних матеріалів; Вартість	1. $30 2. входить у вартість обслуговування 3. безкоштовно 4. входить у вартість обслуговування 5. входить у вартість обслуговування 6. -	1. $150 2. входить у вартість обслуговування 3. безкоштовно 4. входить у вартість обслуговування 5. входить у вартість обслуговування 6. -	1. $60 2. входить у вартість обслуговування 3. безкоштовно 4. входить у вартість обслуговування 5. входить у вартість обслуговування 6. -		1. - 2. - 3. - 4. - 5. - 6. -	1. - 2. вартість експлуатації менша ніж у конкурентів 3. вартість утилізації така ж як у конкурентів 4. вартість витратних матеріалі така ж як у	вартість обслуговування дешевша ніж у конкурентів 1. - 2. - 3. - 4. - 5. присут

	ремонту; Вартість знижки.						конкурентів 5. вартість ремонту така ж як у конкурентів 6. -	ні більш вигідні пропозиції знижки
2.	Призначення (технічні) ОС; Процесор; Пам’ять; Дисковий простір;	1. Linux; 2. SoC Broadcom BCM2837 B0 3. 2Gb; 4. 6 GB;	1. TV OS 2. A10X Fusion chip with 64-bit architecture; 3. 2GB; 4. 32 GB;	1. - 2. NXP iMX6; 3. 256MB; 4. 4 GB;		1. - 2. - 3. - 4. -	1. - 2. - 3. Для нормального функціонування ПЗ необхідне технічне обладнання з пам’яттю як у більшості конкурентів 1. Для нормального функціонування ПЗ необхідне технічне обладнання з дисковим просторо	1. - Підтримує можливість використання процесором з відносно невисокою потужністю 2. - 3. - 4. -

							м як у більшості конкурентів 2. Роздільна здатність екрану така ж як у більшості конкурентів	
3.	Надійність	ПЗ є достатньо надійним для користування, оскільки передбачає захист приватних даних від сторонніх користувачів.	ПЗ є достатньо надійним для користування.	ПЗ є достатньо надійним для користування.		-	ПЗ має незаплямовану репутацію серед майбутніх користувачів, велику увагу при розробці ПЗ покладено для забезпечення надійності від початку функціонування додатку, а всі зусилля направлені на	-

							максимальне відлагодження всього функціоналу ПЗ.	
4.	Функціональність: Керування розкладом; Керування пристроєм власного виробництва; Керування за умовою; Працездатність без інтернет з’єднання;	1. + 2. + 3. + 4. +	1. + 2. + 3. - 4. -	1. + 2. + 3. + 4. -		1. - 2. - 3. - 4. -	1. - 2. - 3. - 4. -	1. - 2. - 3. - Для роботи інтернет з’єднання не потрібне

В результаті дослідження сильних, слабких та нейтральних характеристик ідеї проекту було проаналізовано техніко-економічні характеристики існуючих на ринку конкурентів, що, у свою чергу, надає можливість спрогнозувати частку майбутніх потенційних користувачів. Тож, запропоноване ПЗ вирізняється конкурентоспроможним потенціалом серед перелічених вище характеристик, адже містить близько 9,5% в якості показників слабкої сторони, 47% -

нейтральної сторони, 43,5% - сильної сторони. А, враховуючи, що основна мета майбутнього ПЗ – удосконалення вже існуючих систем з подібним функціоналом, маємо досить успішні наміри завоювати прихильність на ринку.

6.2. Технологічний аудит ідеї проекту

Далі наведений аудит технології, за допомогою якоїможна реалізувати ідею проекту. Визначення технологічноїздійсненності ідеїпроекту передбачає аналіз таких складових (таблиця 6.3):

Таблиця 6.3 – Технологічна здійсненність ідеї проекту.

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1.	Універсальний хаб для розумного будинку «Smart Hub» для керування “розумними” пристпроями на відстані та створення сценаріїв.	NodeJs в середовищі Visual Studio Code. Для можливості з’єднання з пристроями по протоколу mqtt було використано модуль «mosca»	Наявні	Доступні для вільного користування
2.	Створення графічного інтерфейсу програмного додатку.	Мова програмування javascript, фреймворк ReactJs	Наявні	Доступні для вільного користування
3.	З`єднання з пристроями	Mqtt брокер mosquitto	Наявні	Доступні для вільного користування
4.	Планування дій за розкладом.	Мова програмування JavaScript, бібліотека для планування дій “node-cron”	Наявні	Доступні для вільного користування
Обрана технологія реалізації ідеї проекту: веб додаток планується				

розробити мовою програмування NodeJs в середовищі Visual Studio Code з використанням ReactJs для створення GUI мовою програмування JavaScript. Для можливості з'єднання з прістроями буде застосовано mqtt брокер "mosquitto", для можливості графічного відображення – бібліотека "high charts".

6.3. Аналіз ринкових можливостей запуску стартап-проекту

Зробимо аналіз попиту. Далі в таблиці 6.4 наводиться його аналіз, виходячи з наявного ринку.

Таблиця 6.4 – Попередня характеристика потенційного ринку стартап-проекту.

№ п/п	Показники стану ринку (найменування)	Характеристика
1.	Кількість головних гравців, од	3 головних гравців
2.	Загальний обсяг продаж, грн/ум.од	$15,5 млрд
3.	Динаміка ринку (якісна оцінка)	зростає
4.	Наявність обмежень для входу (вказати характер обмежень)	затвердження ліцензійних умов провадження, створення маркетингової стратегії для проведення ефективної рекламної діяльності стосовно ПЗ
5.	Специфічні вимоги до стандартизації та сертифікації	відсутні
6.	Середня норма рентабельності в галузі (або по ринку), %	+15,9%

З таблиці 6.4 можна зробити висновок, що ринок є привабливим для входження за попереднім оцінюванням.

В таблиці 6.5 наведено основні характеристики потенційних клієнтів стартаппроекту.

Таблиця 6.5 – Характеристика потенційних клієнтів стартап-проекту.

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти	Відмінності у поведінці різних потенційних	Вимоги споживачів до товару

		ринку)	цільових груп клієнтів	
1.	Керування “розумними” пристроями.	Власники квартир/офісів	-	Можливість з’єднання пристроїв у одну мережу з можливістю керування пристроями у автоматичному або ручному режимі.

З таблиці 6.5 можна зробити висновок, що цільовою аудиторією є навчальні заклади, компанії, які займаються архітектурою, електричною інженерією, машинобудуванням. Для всіх аудиторій важлива цілодобова технічна підтримка, якість продукції та технічні характеристики.

Проведемо аналіз ринкового середовища та сформулюємо фактори загроз та фактори можливостей у таблицях 6.6 та 6.7.

Таблиця 6.6 – Фактори загроз.

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1.	Недосвідчені учасники команди	Призначення недосвідчених працівників (студентів) для виконання роботи проекту ставить під загрозу дату його завершення, оскільки їм може знадобитися більше часу, щоб ознайомитися з бізнес-моделлю, технологіями.	Щоб мінімізувати цей ризик, необхідно закласти достатньо часу на введення нових працівників у курс справи.
2.	Планування та послідовність виконання задач	Навіть якщо ці завдання виконують різні люди, їх одночасне виконання	Перевірити, чи не заплановано забагато завдань на один і той самий час.

		у великій кількості створює ризик для проекту, особливо наприкінці його реалізації.	При плануванні задач проекту спочатку необхідно скласти список завдань і згрупувати їх, щоб оцінити весь обсяг проекту та кінцеві результати. Потім можна починати зв’язувати завдання, щоб отримати ідеальний розклад.
3.	Потужна клієнтська база конкурентів	Конкуренти, які мають впевнений досвід продукта на ринку здобули сильну базу клієнтів-споживачів.	Розвиток вражаючої маркетингової кампанії, створення стратегії піар-менеджменту, закладання регламенту рекламної кампанії, акційних пропозиції.

Таблиця 6.7 – Фактори можливостей.

№ п/п	Фактор	Зміст	Можлива реакція компанії
1.	Аналіз досвіду конкурентів	Формування стратегії реалізації проекту без навчання на своїх помилках, а при навчанні на помилках конкурентів – невдалі рекламні, маркетингові ходи конкурентів.	Планування і реалізація проекту з максимальним виключенням ймовірності виникнення помилок вже досвідчених конкурентів на ринку споживачів.
2.	Націлення продукту на основні функціональності, які відстутні у конкурентів	Реалізація нових можливостей для споживачів, впровадження покращень суміжного з конкурентами проекту функціональностями.	Чітке планування задач, розподілення задач між розробниками з залученням прорахованих ризиків, мотивація команди ідеєю кінцевого продуту.

3.	Підвищення рентабельності проекту	За рахунок правильного планування всіх етапів проекту, чіткого формулювання бізнес-моделі є можливість залучення до команди проекту студентів в якості розробників.	Зниження кількості інвестицій для розробки і впровадження кінцевого продукту.

Заключним етапом ринкового аналізу, щодо інтеграції продукту є SWOTаналіз. SWOT-аналіз (або SWOT-матриця) - це метод стратегічного планування, який допомагає людині чи організації визначити сильні сторони, слабкі сторони, можливості та загрози, пов'язані з діловими конкурентами чи плануванням проекту. Даний аналіз має на меті визначити цілі підприємницької діяльності або проекту та визначити внутрішні та зовнішні фактори, які є сприятливими та несприятливими для досягнення цих цілей. Користувачі аналізу SWOT часто запитують та відповідають на запитання, щоб створити значущу інформацію для кожної категорії, щоб інструмент був корисним та визначити їх конкурентну перевагу. SWOT був описаний як спрощений і справжній інструмент стратегічного аналізу.

Назва є абревіатурою для чотирьох параметрів, які розглядає техніка.

Сильні сторони: характеристики бізнесу або проекту, що дають йому перевагу перед іншими.

Слабкі сторони: характеристики бізнесу, що ставлять бізнес або проект у невигідному становищі порівняно з іншими.

Можливості: елементи у навколишньому середовищі, які бізнес або проект можуть використати для його переваг.

Загрози: елементи в середовищі, які можуть спричинити проблеми для бізнесу чи проекту.

SWOT – матриця програмно-апаратного комплексу представлена у таблиці 6.8.

Таблиця 6.8 – SWOT- аналіз стартап-проекту

Сильні сторони: Цілодобова технічна підтримка; справка-інструкція по експлуатації; якість продукту; продукт відповідає потребам споживачів; доступність.	Слабкі сторони: Низька репутація компанії на початку впровадження проекту в життя; присутність багів.
Можливості: Вихід на міжнародний ринок; результативність; розвиток нових функціональних можливостей.	Загрози: Зниження доходів потенційних клієнтів; блокування реклами на просторах інтеренету, соцільних мереж; блокування інтернет-ресурсу програмного забезпечення.

З SWOT- аналізу стартап-проекту можна зробити висновок, що сильними сторонами є цілодобова підтримка, інструкція по експлуатації, якість продукту, відповідність потребам споживачів та доступність. А слабкими сторонами є низька репутація компанії на початку впровадження проекту в життя та присутність багів.

ВИСНОВКИ

Розроблено веб-середовище для моделювання процесів міжагентної взаємодії в умовах “розумного будинку”.

Даний продукт написаний на мові JavaScript з використанням GraphQL та Apollo. Завдяки такому підходу, вдалося досягти поставлених цілей, таких як легка підтримка системи, кросплатформність, можливість її масштабування, доповнення додактовими модулями та функціоналом. Цьому також сприяє використання мікросервісного подходу до проектування. В даній програмі використовувались всі новітні можливості мови JavaScript такі як: проміси, модулі, класи, ітератори.

Даний додаток може входити до різних систем, які призначені для автоматизації приміщень. Розроблений програмний додаток може використовуватися у програмних системах для різних платформ.

Хоча розроблена тестова схема і відображає малий набір даних, проте з точки зору моделі, яку вона відображає – робоча модель має великий потенціал до розширення. Всі датчики є абстрактним уявленням більш складних приладів, які з легкістю можуть бути впроваджені в систему, розширюючи її до потреб користувача. Були продемонстровані всі типи керування в обидва напрямки: датчики-сервер, сервер-датчики.

Результати даної роботи можуть бути використані для розгортання системи з контролем кліматичних параметрів і енергоспоживання як в звичайному будинку, так і в спеціалізованих приміщеннях з більш жорсткими вимогами до мікроклімату середовища (серверні, аранжереі, тощо).

-

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Design Patterns, Elements of Reusable Object-Oriented Software / E.Gamma, R. Helm, R. Johnson, J. Vlissides. — Boston: Addison–Wesley, 1994. — 395 с.

2. Docs | Node.js [Електронний ресурс]. – Режим доступу: https://nodejs. org/en/docs. – 02.07.2019.

3. Tutorial: Intro To React - React [Електронний ресурс]. – Режим доступу: https://facebook.github.io/react/tutorial/tutorial.html. – 02.06.2019.

4. Офіційна сторінка Express для NodeJs — Режим доступа: http://expressjs.com/uk/ — Дата доступа: 06.06.2019.

5. Стефанов С. React.js. Быстрый старт. СПб.: Питер, 2017, 304 с

6. Умный дом – Режим доступу : http://www.directinfo.net/index.php? option=com_content&view=article&id=139%3 A2010-07-06-13-57-09&catid=1%3A2008-11-27-09-05-45&Itemid=84&lang=ru - Дата доступу : 14.08.2019.

7. Система умный дом – технология экономии, удобства и комфорта высокого уровня. – Режим доступу : http://smarton.com.ua/smart_home/systema_umniy_dom_intro - Дата доступу : 14.08.2019.

8. Система умный дом. – Режим доступу : http://intelcity.com.ua/comfort_house - Дата доступу : 14.09.2019.

9. Client-Server Interaction over the Internet – Режим доступу: http://www.icodeguru.com/dotnet/core.c.sharp.and.dot.net/0131472275/ ch16lev1sec1 .html. – Дата доступу: 12.09.2019.

10. Mongoose Api Docs – Режим доступу: http://mongoosejs.com/docs/api.html. – Дата доступу: 10.09.2019.

11. ES-2015 сейчас [Електронний ресурс]. – Режим доступу: https://learn. javascript.ru/es-modern-usage. – 04.09.2019.

12. Сайт MQTT — Режим доступу: http://mqtt.org/ — Дата доступа: 05.09.2019.

13. Martin Fowler – Microservices – Режим доступу: http://martinfowler.com/articles/microservices.html – Дата доступу: 28.09.2019.

14. Принцип MVC в веб программировании. – Режим доступу: http://folkprog.net/printsip-mvc-u-web-programmirovanii/. – Дата доступу: 23.09.2019.

15. REST - Semantic Web Standards - W3C. – Режим доступу: https://www.w3.org/2001/sw/wiki/REST. – Дата доступу: 6.09.2019.

16. Hypertext Transfer Protocol overview. – Режим доступу: https://www.w3.org/Protocols/. – Дата доступу: 18.09.2019.

17. Richardson, Leonard. RESTful Web APIs. / Richardson, Leonard. // O'Reilly Media, 2013 – pp. 74 – 79.

18. NPM official website. - Режим доступу: https://docs.npmjs.com. - Дата доступу: 28.10.2019.

19. PubSubClient Github page. - Режим доступу: https://github.com/knolleary/pubsubclient. - Дата доступу: 28.10.2019.

20. В.Н. Гололобов. «Умный дом» своими руками. / В.Н. Гололобов – М.: НТ Пресс, 2007. – 416 с.

21. Офіційний сайт компанії Google — Режим доступ: https://cloud.google.com/solutions/iot/ — Дата доступа: 04.10.2019р

22. KhaosT/HAP-NodeJS: Node.js implementation of HomeKit Accessory Server. – Режим доступу: https://github.com/KhaosT/HAP-NodeJS - Дата доступу: 28.11.2019.

23. apollo - npm – Режим доступу: https://www.npmjs.com/package/apollo - Дата доступу: 28.11.2019.

24. GraphQL | A query language for your API – Режим доступу: https://graphql.org/ - Дата доступу: 28.11.2019.
25. Implementing RabbitMQ With Node.js - Better Programming – Medium – Режим доступу: https://medium.com/better-programming/implementing-rabbitmq-with-node-js-93e15a44a9cc/ - Дата доступу: 22.11.2019.
26. Building Microservices Architecture With Node.Js. - Data Driven Investor - Medium – Режим доступу: https://medium.com/datadriveninvestor/building-microservices-architecture-with-node-js-caf3a67ed6bd - Дата доступу: 24.11.2019.
27. HomeKit Accessory Protocol Specification (Non-Commercial Version) - Support - Apple Developer – Режим доступу: https://developer.apple.com/support/homekit-accessory-protocol/ - Дата доступу: 25.11.2019.
28. 256dpi/arduino-mqtt: MQTT library for Arduino – Режим доступу: https://github.com/256dpi/arduino-mqtt - Дата доступу: 26.11.2019.
29. Docker Hub - Container Image Library | Docker Arduino – Режим доступу: https://www.docker.com/products/docker-hub - Дата доступу: 27.11.2019.
30. Overview of Docker Compose – Режим доступу: https://docs.docker.com/compose/ - Дата доступу: 21.11.2019.
31. An Introduction to Environment Variables and How to Use Them – Режим доступу: https://medium.com/chingu/an-introduction-to-environment-variables-and-how-to-use-them-f602f66d15fa - Дата доступу: 23.11.2019.
32. Subscriptions - Client (React) - Apollo GraphQL Docs – Режим доступу: https://www.apollographql.com/docs/react/data/subscriptions/ - Дата доступу: 28.11.2019.

Додаток 1

Моделювання процесів міжагентної взаємодії в мережах Smart Grid

Апробації

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР4164_18М

Аркушів 1

2019

УДК 621.43.056:632.15

Магістрант 5 курсу, гр. ТР-81мп Швайка Д.А.

Доц., к.ф.-м.н. Тарнавський Ю.А.

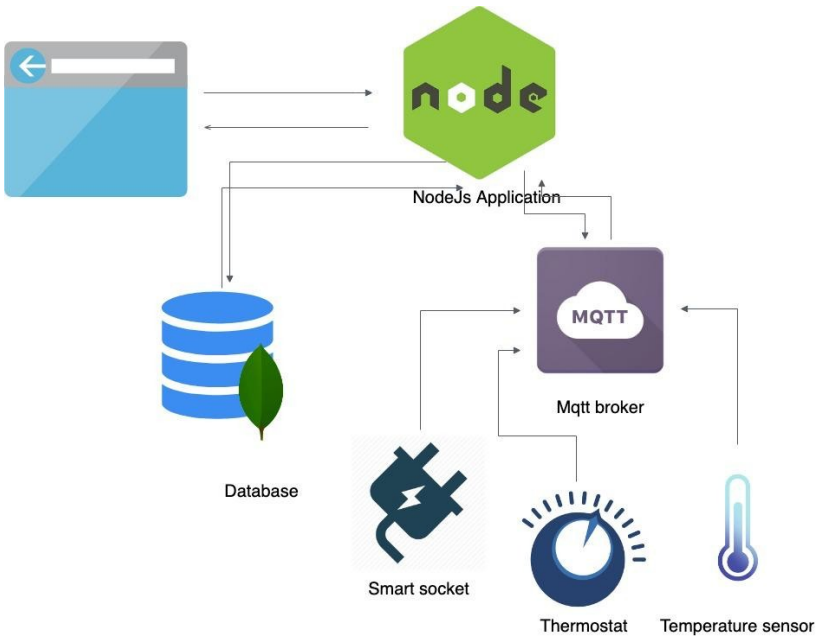
ВЕБ-СЕРЕДОВИЩЕ ДЛЯ МОДЕЛЮВАННЯ ПРОЦЕСІВ МІЖАГЕНТНОЇ ВЗАЄМОДІЇ В МЕРЕЖАХ SMART GRID

Будь-який будинок - адміністративний, виробничий або житловий складається з деякого набору підсистем, що відповідають за виконання певних функцій, які вирішують різні завдання в процесі функціонування цієї будівлі. Сучасна будівля такого типу - це місто в мініатюрі. Тому досить актуальною стала ідея автоматизування певних дій в таких будівлях.

Для цього були використані технології веб-розробки на мові JavaScript, а саме фреймворк ExpressJs. Основна ідея фреймворку полягає у наданні розробнику усіх необхідних модулів. Це робить розробку веб застосунку дуже простою не вимагаючи багато часу на підготовку.

Особливістю програмної системе є те, що вона дає можливість автоматизувати дії у будинку за допомогою мережевих модулів: сенсорів, розумних вимикачів, реле і т.ін. Система не потребує налаштування мереживих модулів, одразу після з'єднання вони починають працювати.

Розроблена система складається з nodeJs додатку, у якості бази даних було використано mongoDb та mqtt broker. Веб інтерфейс розроблений за допомогою js фреймворку ReactJs.



Архітектура розробленої системи

Перелік посилань:

1. М. Э. Сопер. Практические советы и решения по созданию « Умного дома » » / М. Э. Сопер. – М.: НТ Пресс, 2007. – 432 с.

2. Амосов А. А. Вычислительные методы для инженеров / А. А. Амосов, Ю. А. Дубинский, Н. П. Копченова. — М:Мир, 1998. – 644 с.

3. В.Н. Гололобов. «Умный дом» своими руками./ В.Н. Гололобов – М.: НТ Пресс, 2007. – 416 с